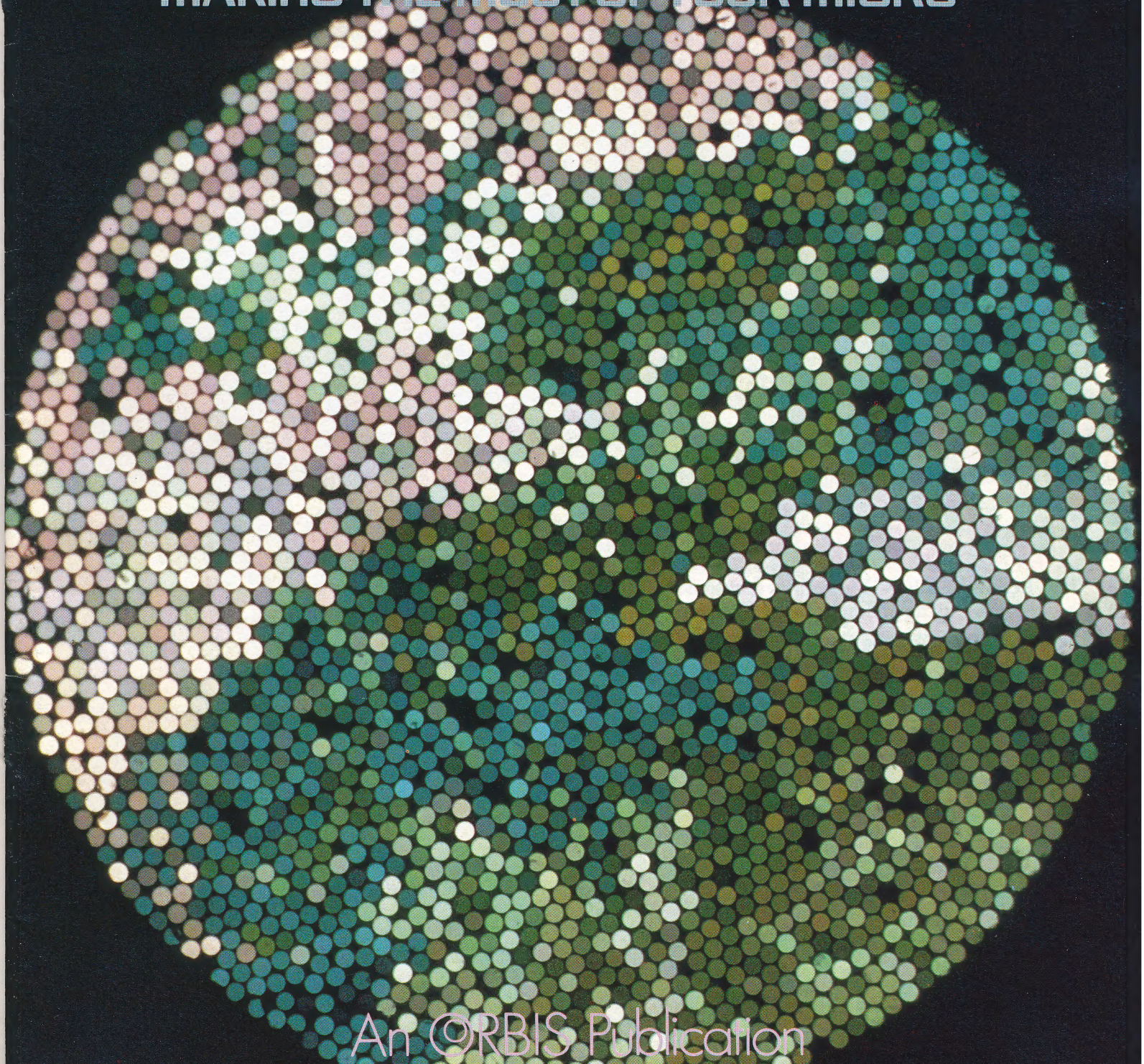
ISSN 0265-2919

90p 76

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£ 1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

INDUSTRIAL ACTION A short series begins on computers in industry 1510

HARDWARE

THE LIGHT FANTASTIC We shed some light on optical computer technology 1501

SOFTWARE

WHO, WHERE AND HOW Our interactive characters are ready to be given some meaning in their lives 1506

RELAX, DO IT Frankie Goes To Hollywood comes on disk and cassette 1515

COMPUTER SCIENCE

A PRIME EXAMPLE FORTH's arithmetical abilities are highlighted with an 'Eratosthenes' Sieve' program 1504

JARGON

FROM RING NETWORK TO ROUNDOFF ERROR A weekly glossary of computing terms 1509

PROGRAMMING PROJECTS

GROUP AWARENESS Our Go program will now deal with all possible groupings of stones 1512

MACHINE CODE

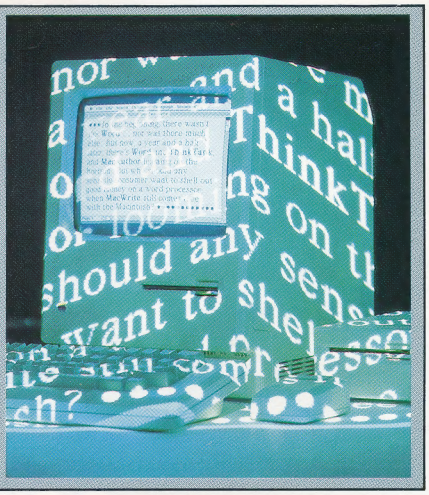
PICTURE PERFECT Graphics and screen handling using the Spectrum's OS ROM 1518

WORKSHOP

METERING REQUIREMENTS We continue our look at the 7135 chip 1516

Next Week

- **We begin a new machine code series in which we take an in-depth look at the Amstrad operating system**
- **The Apple Macintosh has gained the company a reputation for innovation. Apple has recently launched the LaserWriter, a printer that produces text by means of a laser beam. We throw some light on this device**
- **Computer aided design (CAD) is becoming one of the most important aspects of computers in industry. We examine some of the techniques and products**



QUIZ

1) What device is the optical equivalent of the transistor?
2) What is the difference between the display file and the attributes file on the Spectrum?
3) What do the letters AMT stand for?

Answers To Last Week's Quiz

1) The Transient Program Area (TPA) on the Amstrad is not sufficiently large on the Amstrad to enable WordStar to run properly.
2) A hierarchical database finds its information by travelling down a series of 'paths' until it reaches the required data, whereas a relational database obtains information by a mathematical formula whose result will obtain the information directly.

Coming Up. . .

- A series on programming the 68000 microprocessor, the chip at the heart of the new Atari ST range and the Apple Macintosh
- A close look the MS-DOS industry standard operating system, now implemented on the majority of business micros, and its close relations, MSX-DOS and PC-DOS

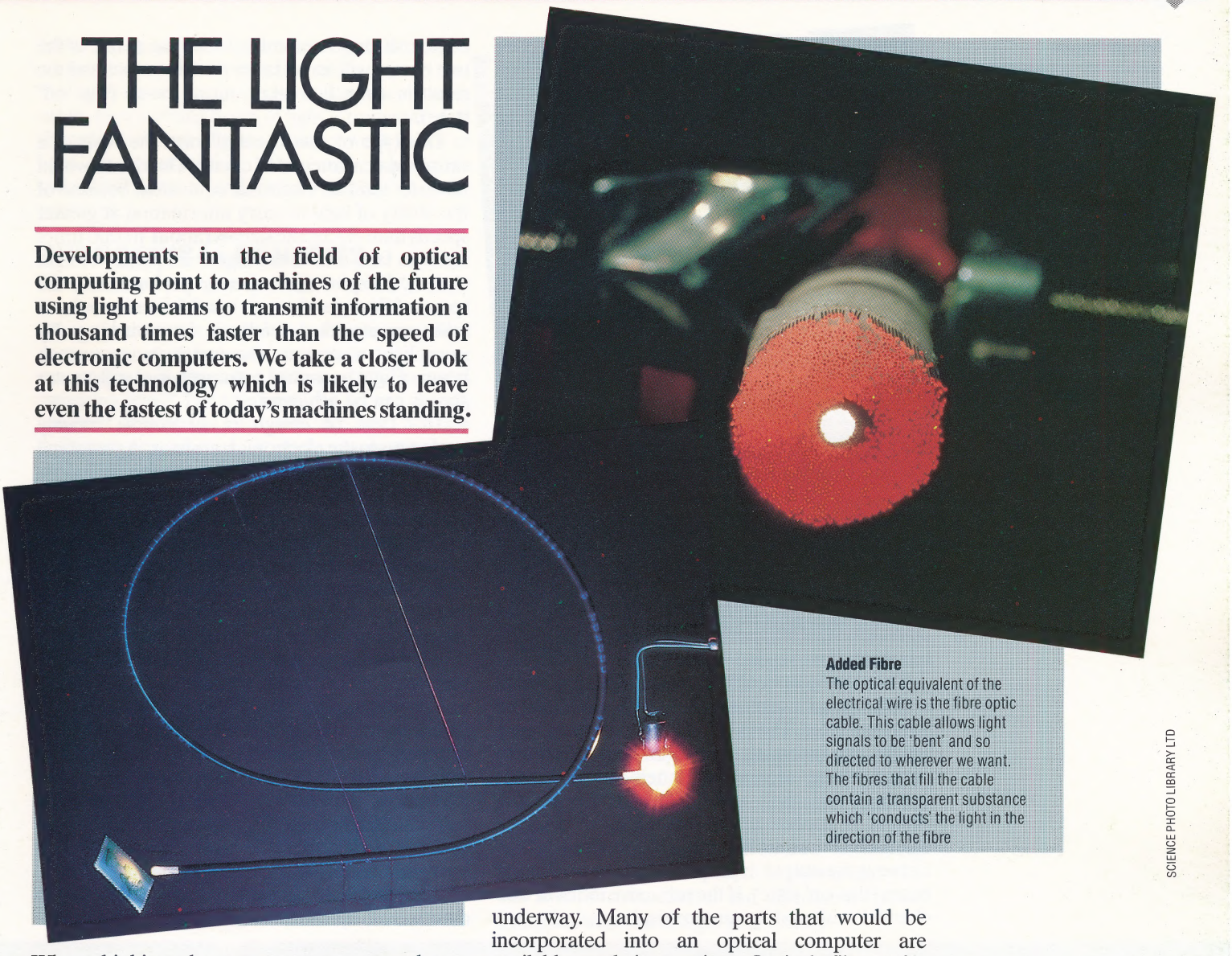
Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editors** Jon Kaye, Nik Lumsden; **Contributors** Dr N M Rufford, Steve Cooke, Helena Sturridge, Caroline Clayton, Marcus Jeffery, Steve Colwill, Steve Malone, Steven Vickers, David Mudd; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

Our cover shows a close-up of a fibre optics cable head, magnified approximately 12 times to show the individual fibres. Cables like this one are capable of transmitting data at very high speeds and offer clear advantages over their older electronic counterparts SCIENCE PHOTO LIBRARY LTD

THE LIGHT FANTASTIC

Developments in the field of optical computing point to machines of the future using light beams to transmit information a thousand times faster than the speed of electronic computers. We take a closer look at this technology which is likely to leave even the fastest of today's machines standing.



Added Fibre
The optical equivalent of the electrical wire is the fibre optic cable. This cable allows light signals to be 'bent' and so directed to wherever we want. The fibres that fill the cable contain a transparent substance which 'conducts' the light in the direction of the fibre

SCIENCE PHOTO LIBRARY LTD

When thinking about a computer, you almost certainly envisage a plug-in device, based on electronic binary switches which store and process information. You would be wrong, however, to conclude that this is the only feasible type of computer. A computer could conceivably be based on any method of storing information in physical form: electrical charge, teeth and gaps in gear wheels, holes in pieces of paper, beads, pebbles, seashells or just about any other media.

The electronic semiconductor is the best type of information store so far invented because it consumes hardly any power, it is of microscopic size and, most important of all, it is fast. The speed at which a computer can operate is limited by the time required for its binary information stores, or 'switches', to change from one position to another and by the speed of transmission of information between switches. While computer circuits based on streams of electrons are very fast at performing these functions, it is theoretically and technically possible to build circuits which are faster still, based on beams of light.

Research into optical computing and related disciplines, like 'photonic logic', is already underway. Many of the parts that would be incorporated into an optical computer are available and in service. Optical fibres, for example, are being used to carry telephone and television signals. Optical waveguides, the equivalent of the metal conductors on printed circuit boards, have been developed. So have optical disks. The most crucial component of all, the optical analogue of the electronic binary switch, is currently on the test bench.

Designs for an optical binary switch fall into two principal types: electro-optical switches and all-optical switches. It is worth discussing each of the two types, starting with the electro-optical hybrid, since optical components will probably be used first in conjunction with electronic components rather than in competition with them. Both types of switches exploit the phenomenon of light interference caused by controlled changes in the refractive index of a non-linear optical material, usually a compound of gallium or lithium. The refractive index of a material expresses the ratio of the speed of light in a vacuum to its speed in the material. For example, if the refractive index of a particular type of glass is 1.5, then, whereas light travels at about 300,000 kilometres per second in a vacuum, its speed will be about 200,000

The Light Program
Another important component that will be necessary if the optical computer is to be realised is this semiconductor device, which can be tuned to emit a range of pure light frequencies. This will be vital for communication interference purposes where light on a standard frequency and wavelength is required

SCIENCE PHOTO LIBRARY LTD

kilometres per second in the glass.

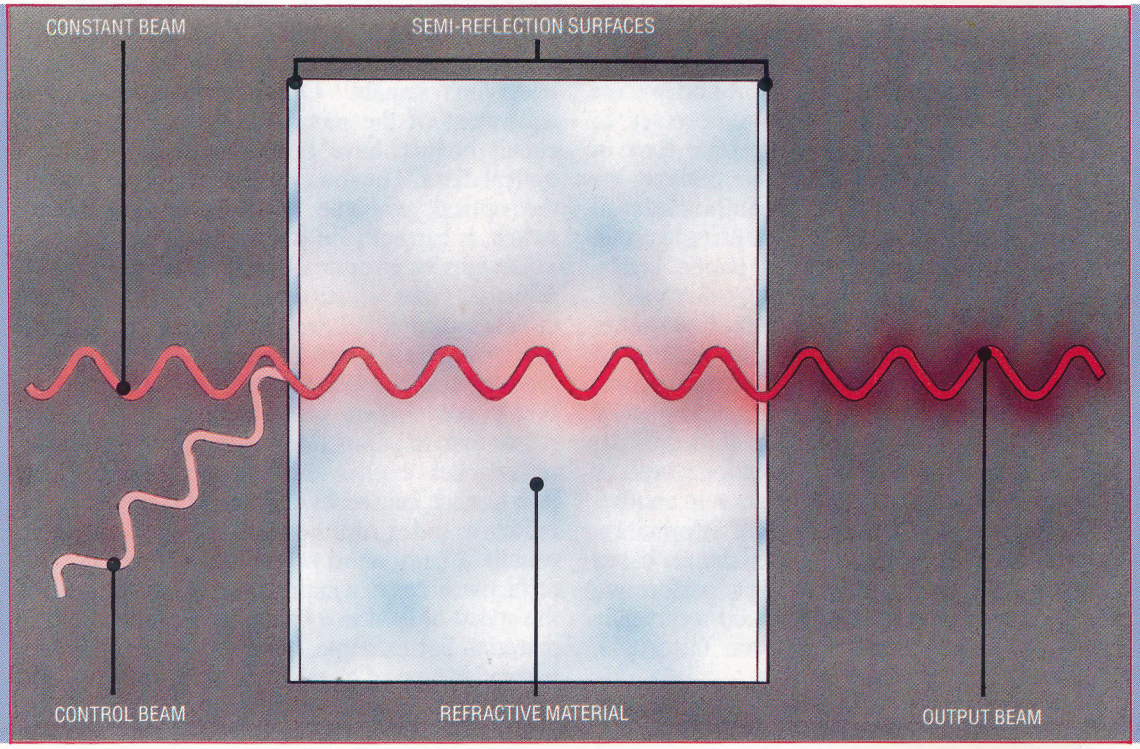
The electro-optical hybrid type of binary switch is based on an invention called the Mach Zehnder interferometer. The principle of this device is very simple. An incoming light beam is split into two, passed along separate channels of non-linear material and then recombined. If no electric current is applied across one or another of the channels of the interferometer, the separate beams arrive at the output in phase and generate a strong beam (the 'on' state). If the refractive index of one channel is altered by an electrical signal, the two beams no longer recombine in phase, as one of the two beams will travel faster than the other and the result is a much weaker output beam (the 'off' state).

This type of switch could form the basis of a simple optical integrated circuit. The circuit would be faster than an electronic equivalent because of the ability of light to carry information at greater speeds than electronic signals (about 10,000 times faster in fact, due to the higher frequency of light waves). Despite this superior performance, the speed of this hybrid circuit is still limited because it relies on electrical feedback for switching. By using light to operate the switch (using a weak beam of light to control a stronger one) still greater speeds can be achieved.

This type of design is, of course, exactly analogous to the electronic transistor. A prototype version of an all-optical switch, called a *transphasor*, has already been built by a team at Heriot Watt University, Edinburgh. The device is capable of switching in about a picosecond (a thousand billionth of a second). The fastest electronic transistor, by comparison, has a switching speed of a nanosecond (a billionth of a second).This means that an all-optical computer could potentially operate at speeds 1,000 times greater than an electronic computer.

The transphasor developed at Heriot Watt is based on the design of an interferometer first invented by the French physicists Charles Fabry and Alfred Perot in 1896. A sliver of non-linear material is sandwiched between two reflecting surfaces. Light entering through one surface bounces back and forth inside the sandwiched layer before leaving through the opposite surface. The trapped light waves interact in a way that is determined by the refractive index of the non-

On The Beam
The basic building block of the electronic computer is the transistor, and its optical equivalent is known as the 'transphasor'. A constant beam is input at one end together with a 'control beam' which will interfere together, within the device. A small variation in the intensity of the control beam will produce a large change in the intensity of the output light. Thus the transphasor can be used as a logical switching device



KEVIN JONES

linear material, which can be adjusted using a control beam. If the refractive index is such that the light waves are in phase and reinforce each other, then a strong beam of light is transmitted. If the waves are out of phase, only a weak beam of light is transmitted. The two alternative transmission states correspond to the 'on' and 'off' positions.

THEORETICAL BENEFITS

The success of the optical computer will ultimately be determined by the theoretical and technical limits of conventional machines. On paper, optical computers are superior to their electronic counterparts in a number of respects. The first is switching speed. For a transistor to act as a switch, a current of electrons must cross its base. There are limits to the speed at which electrons can move in a semiconductor, and limits to how thin a transistor base can be made (the thinner the base, of course, the shorter the journey for the electrons). An optical computer, transmitting at the speed of light, is inherently much faster.

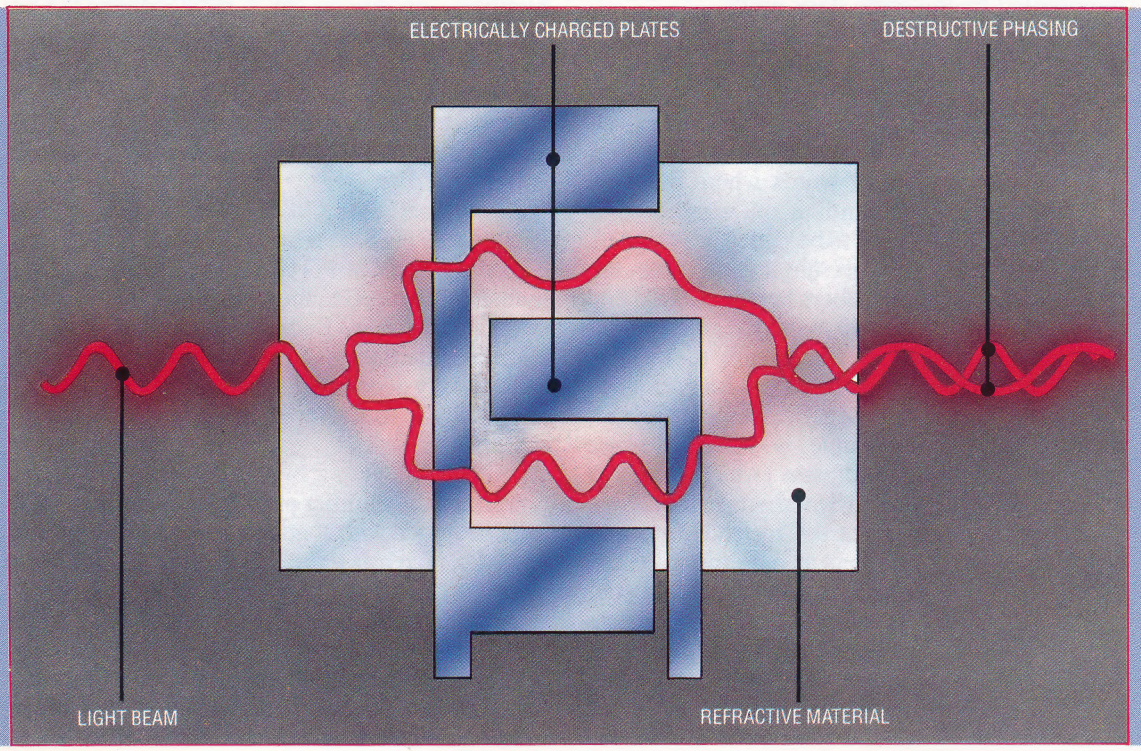
Conventional machines are also restricted by something called 'the von Neumann bottleneck'. This is a feature of the basic architecture of nearly all computers and concerns the way in which information is accessed from the memory a word at a time (in 'serial' fashion). In an optical computer, memory could be accessed in parallel. This is because light beams do not need to insulated in the same way as electrical signals. Separate beams could therefore be channelled through the same optical transistor and switched simultaneously.

The third constraint of conventional computer design is the limited space available for wiring on a flat chip. Currently, the number of connections that can be made to a single chip is severely constrained because of the 'pin fanout' problem — a limit on the number of pins that can be mounted on a flat piece of silicon. Using three-dimensional (optical) holographic techniques, it would be possible in principle to introduce an extra degree of freedom for wiring, and to instantly change the wiring configuration.

Finally, there are practical benefits in the interconnection of optical computers and peripherals. Optical fibre networks are already used for linking electronic devices. They are extremely rapid and light messages can be transmitted on the same optical fibre without interference. Optical computers could be linked together by these networks without the need for slow opto-electrical interfaces.

Given these advantages, how soon can we expect practical and affordable optical computers to start appearing? The answer is: not for some time. The theoretical benefits of optics as compared to electronics are not the same as the practicalities. Massive strides in chip miniaturisation mean that, in the short term at least, optical computers are chasing a constantly receding target. The number of transistor devices that can be packed on to a chip is still doubling every eighteen months. Chips with 10 or 100 million devices are predicted by the year 2000. Large scale integration (LSI) has developed into very (and ultra) large scale integration (see page 194). The 'parallel processor' (see *parallel processing*, page 1228) has grown into the 'massively parallel processor'.

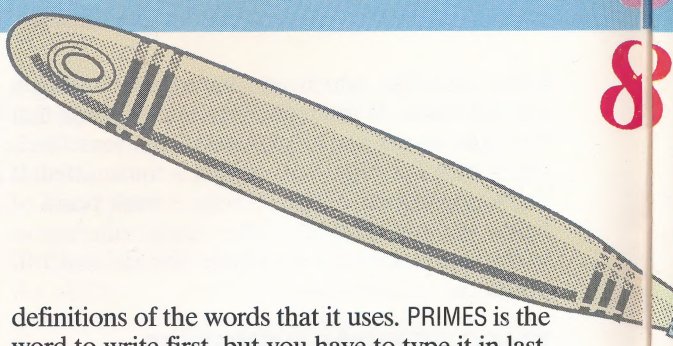
The latest high-speed electronic technology is continuing to develop and improve, and with this sort of competition, the affordable optical computer may still be light years away.



KEVIN JONES

Face To Phase
The Mach Zehnder interferometer enables electronic and optical devices to work together. Light entering at one end is split into two beams. Once the beams have entered the device (by adjusting the refractive index around one of the beams), the speed of the light will be slowed down and will move 'out of phase' with the other beam. When the two beams converge in the output beam, the phases will interfere destructively and the result will be 'off', which can be considered at logical zero

A PRIME EXAMPLE



FORTH deals with numbers using fixed point, and integer arithmetic. We illustrate this approach with a listing of the FORTH version of Eratosthenes' Sieve, which can be compared to the PASCAL version we looked at on page 1148. By so doing, you can better understand FORTH's arithmetical syntax.

One of the features of FORTH that can be puzzling at first is its restriction to integer arithmetic. This is something that is not suggested by interactiveness and extendability but by efficiency: integer arithmetic is much faster than floating point.

The usual way of handling fractions in FORTH is with *fixed point* arithmetic, in which you use numbers that are multiplied or divided by some constant, usually a power of ten. This is like using millimetres when metres aren't accurate enough.

A number on the FORTH stack has 16 bits, and so can be considered as either a *signed* integer (the sign being indicated by the most significant bit) between −32768 and +32767 or an *unsigned* integer between 0 and 65535. The quickest way to see this ambiguity is by typing:

```
65535 .
```

This displays the answer −1, because . treats the top of the stack as a signed number, and the unsigned number 65535 and the signed number −1 are represented by the same stack entry.

The FORTH word U. is just like . except that it treats the top of the stack as unsigned. Thus:

```
65535 U.
```

displays 65535.

There are some other words where there are different versions, depending on whether they expect the stack numbers to be signed or unsigned. The choice becomes important if there is ever the possibility of getting numbers between 32768 and 65535. You have to decide to use the words either for signed numbers, in which case these large numbers count as overflow, or for unsigned numbers, in which case negative numbers are not allowed. A good example is when you use numbers as memory addresses (which is how @ and ! work). These are unsigned, so you may have to use unsigned words, like U< instead of <. You can also use stack numbers as patterns of 16 bits and do the usual bitwise Boolean combinations.

The main example shows a program that uses the Sieve of Eratosthenes to display all primes up to 65536. As is common with FORTH programs, you'll find it easier to read it backwards: start at the main word, PRIMES, and look back to find the definitions of the words that it uses. PRIMES is the word to write first, but you have to type it in last, after its subwords.

For the Eratosthenes' Sieve program, you write down all the numbers up to a specified limit. When you find a prime, you cross out all its multiples, which obviously can't themselves be prime, and the next prime is the next number that hasn't been crossed out. Ignoring 1 (which, for various reasons, isn't considered a proper prime) you find the first prime, 2, and cross out all its multiples. The next number that hasn't been crossed out, 3, is the next prime, so you cross out all its multiples and so on. You can easily do this with pencil and paper to find all primes up to 20.

The FORTH program listed here stores eight Kbytes or 65,536 bits—one for each number up to 65536. They all start off at 0, but when a number is crossed out, its bit is changed to 1. The memory space for these bytes is set up by:

```
CREATE BITS 8192 ALLOT
```

(We have actually replaced 8192 by a constant, BYTES.) We shall see later more precisely how this works, but for now its effect is to allocate a block of 8,192 bytes somewhere in memory and create a new word, BITS, whose action is to stack the address of the first of these bytes. It can thus be used to contain an array of bytes.

To refer to one of the bits, you need two numbers—one shows which byte contains the bit (so for this number you can use the memory address of the byte), and the other shows which bit it is within that byte, so it is between 0 and 7. It's not too hard to translate between this pair of numbers and the number (between 1 and 65536) that the bit is for.

Remember that CURBYTES is a variable; its value, CURBYTES @, is an address of a byte in the BITS array, and CURBYTES @ C@ is the value of the byte. If you're used to the C language, this idea will be quite familiar. Because the prime and the addresses are unsigned and possibly quite big, we occasionally need unsigned words like U. and U<.

Notice the useful word:

```
+! (n, address – )
```

It replaces the number at the given address by the same number with n added on, so it is typically used for increasing the value of a variable by a given amount.

A final point about arithmetic is that some words are for *double length* arithmetic. This means that two two-byte numbers on the stack are used together to make a single four-byte number. The more significant half is nearer the top of the stack,

Number Bases

It is very easy in FORTH to switch number bases, so that numbers are printed or read from the keyboard in binary, octal, decimal, hex or any other number base you choose. You do this by changing the variable BASE. For instance:

```
54 16 BASE ! . DECIMAL
```

This displays 36 (54 in hex), and switches the number base back to ten with the word DECIMAL.

In practice you'd find it easier to define words like

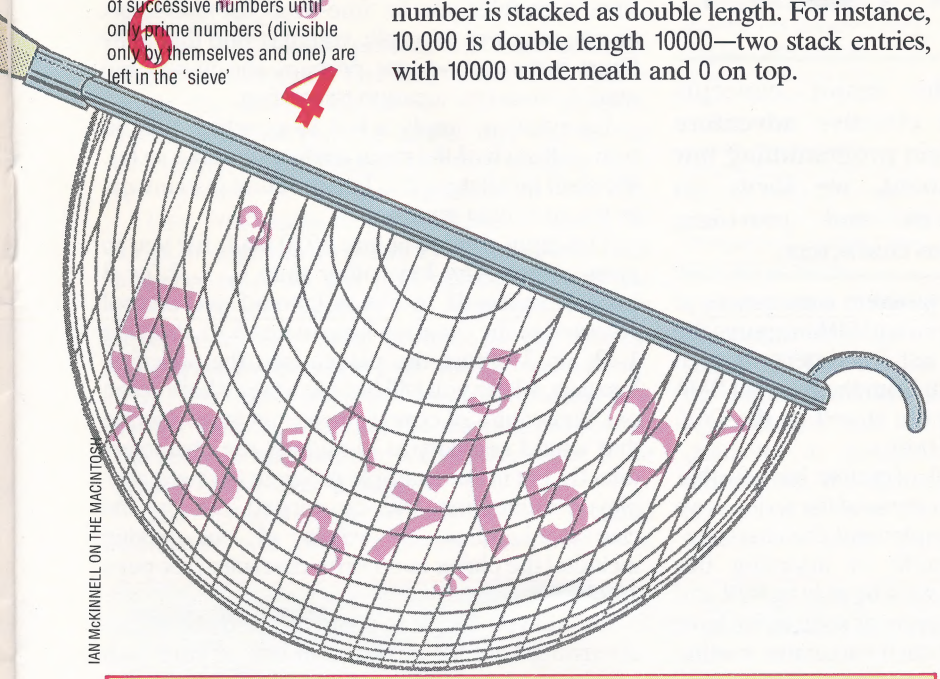
```
: HEX 16 BASE ! ;
```

Note that once a number (like 16 here) is part of a word definition, it is not affected by later changes to the number base.

Prime Selection
Eratosthenes' Sieve, named after the Greek mathematician who formulated it, is a simple but effective method of finding prime numbers. The method works by 'sieving out' multiples of successive numbers until only prime numbers (divisible only by themselves and one) are left in the 'sieve'

the less significant half underneath.

Although the FORTH-83 standard only allows you to type in single length numbers, in most implementations if a number contains a full stop (at any point within the number) then that is ignored (it's not treated as a decimal point) and the number is stacked as double length. For instance, 10.000 is double length 10000—two stack entries, with 10000 underneath and 0 on top.



IAN McKINNELL ON THE MACINTOSH

```
8192  CONSTANT  BYTES
CREATE BITS BYTES ALLOT BITS BYTES
+ CONSTANT BITSEND
(  1 bit for each number 1 - 65536 )
VARIABLE  CURBYTE   VARIABLE  CURBIT
(  show number being looked at.  CURBYTE )
(  points byte in BITS, CURBIT to bit  )
VARIABLE  MASK  (  2**CURBIT  )
VARIABLE  PRIME  VARIABLE  PRIME/8
VARIABLE  PRIMEMOD8
(  show prime when found  )
:  SETUP  (  --  )  ( initialises variables )
   BYTES  0  DO  ( zero array BITS )
      0  BITS  I  +  C!
   LOOP
   BITS  CURBYTE  !  0  CURBIT  !  1  MASK  !
;
:  NEXTBIT  (  -- 0 or non-zero  )
(  adjusts CURBYTE etc to find next bit  )
(  and stacks it  )
   1  CURBIT  +!
   MASK  @  2*  MASK  !
   CURBIT  @  8  =  IF
      0  CURBIT  !
      1  MASK  !
      1  CURBYTE  +!
   THEN
   CURBYTE  @  C@  MASK  @  AND
;
:  PRIMEETC  ( -- )  ( sets up PRIME, PRIME/8 )
(  and PRIMEMOD8 )
   CURBIT  @  1+
   DUP  8  =  IF
      DROP  0  PRIMEMOD8  !  1
   ELSE
      PRIMEMOD8  !  0
   THEN
   CURBYTE  @  BITS  -  +  PRIME/8  !
   PRIME/8  @  8  *  PRIMEMOD8  @  +  PRIME  !
;
:  ANOTHER-PRIME  (  --  0 or -1  )
(  adjusts CURBYTE etc to next prime, sets  )
(  PRIME etc. 0 left if end of BITS reached  )
   BEGIN
   NEXTBIT  0=  UNTIL
   CURBYTE  @  BITSEND  U<  IF
      PRIMEETC
      -1
   ELSE
      0
   THEN
;
VARIABLE  DELBYTE  VARIABLE  DELBIT
:  DELMASK  (  --  2**DELBIT  )
   1
   DELBIT  @  IF   (  if DELBIT non-zero  )
      DELBIT  @  0  DO  (  multiply 1 by 2  )
         2*             (  2 DELBIT times  )
      LOOP
   THEN
;
:  ANOTHER-MULTIPLE  (  --  0 or -1  )
(  adjusts DELBYTE and DELBIT  )
(  to next bit if set  )
(  0 left if end of BITS array reached  )
   PRIMEMOD8  @  DELBIT  +!
   PRIME/8  @  DELBYTE  +!
   DELBIT  @  7  >  IF
      -8  DELBIT  +!
      1  DELBYTE  +!
   THEN
   DELBYTE @ BITSEND U< BITS
   1- DELBYTE @ U< AND
(  careful in case DELBYTE exceeds 65535  )
;
:  DELETE-MULTIPLES  (  --  )
(  sets bits for m/tiples of current prime  )
   CURBYTE  @  DELBYTE  !
   CURBIT  @  DELBIT  !
   BEGIN
   ANOTHER-MULTIPLE  WHILE
    DELBYTE  @  C@  DELMASK  OR  DELBYTE  @  C!
   REPEAT
;
:  PRIMES  (  --  )
(  prints all primes between 2 and 65535  )
   SETUP
   BEGIN
   ANOTHER-PRIME WHILE
      PRIME @ U.
      DELETE-MULTIPLES
   REPEAT
;
```

Arithmetic Words

Addition and subtraction: there is no distinction between signed and unsigned. + and – are single length, D+ and D– double length.

Unary minus (negating just one number): NEGATE for single length, DNEGATE for double length. figFORTH uses MINUS and DMINUS.

Multiplication: * is single length, signed or unsigned. UM* is unsigned 'mixed' length – it takes two single length unsigned numbers off the stack and leaves their double length product. In older FORTHs UM* is called U*, and there may also be a signed mixed length product M*.

Division: /, MOD and /MOD are single length signed. They leave the quotient, the remainder and both (quotient on top). UM/MOD is an unsigned mixed length version of /MOD . Its dividend (the number you divide into) is double length. In older FORTHs this is called U/ or U/MOD .

There is a difference between FORTH-83 and older FORTHs in how they divide negative numbers. If you imagine doing the division exactly and then rounding to some integer, then FORTH-83 rounds to an integer that is no bigger. e.g. 10/3 rounds down to 3, but –10/3 rounds down to –4. The older FORTHs round towards 0, so that –10/3 rounds up to –3. Once the quotient has been calculated by these rules, the remainder is determined by the formula:

dividend = divisor * quotient + remainder

Multiplication combined with division: */ is signed single length, with the stack effect:

n1, n2, n3 –– n1*n2/n3

Comparisons: = < > and U< are single length. < and > are signed, U< is unsigned, = is either. D= D< and DU< are double length. D< is signed, DU< unsigned and D= either. MAX and MIN (giving the larger or smaller of two numbers) are single length signed. DMAX and DMIN are double length signed.

Absolute value: ABS and DABS are single and double length.

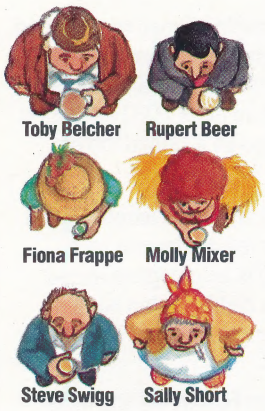
Bitwise Boolean operations: AND, OR, XOR and NOT are all single length. The sign doesn't matter.

Standard combinations: some words are so commonly used together that their combination, without a space, is specially defined as a single word. For instance, 1+ has exactly the same effect as 1 + (although it may be more efficiently done).

Printing: . and U. are single length, respectively signed and unsigned. D. is double length signed.

Memory addressing: an address, single length unsigned, may be referring to the two bytes at that address (as with @ and !), just one byte there (for which you use C@ and C! – C stands for 'character') or the four-byte double length number there (use 2@ and 2!)

WHO, WHERE AND HOW



FIGURES BY DAVID HIGHAM

Opening Time
Our diagram shows the initial positions of the different characters as set by the default vaues in lines 6030 and 6040. Objects, whose data is held in lines 6140 and 6150, are also shown. The player starts the game in the lounge

Having discussed the major concepts behind designing an effective adventure game, we can now begin programming our own. In this instalment, we focus on initialising the screen and providing attributes for the various characters.

The first module in our program commences at line 10, initialising the screen and DIMensioning the arrays discussed in the last instalment. It then reads the necessary data in from the relevant DATA statements, all of which are stored in the DATA module, starting at line 6000.

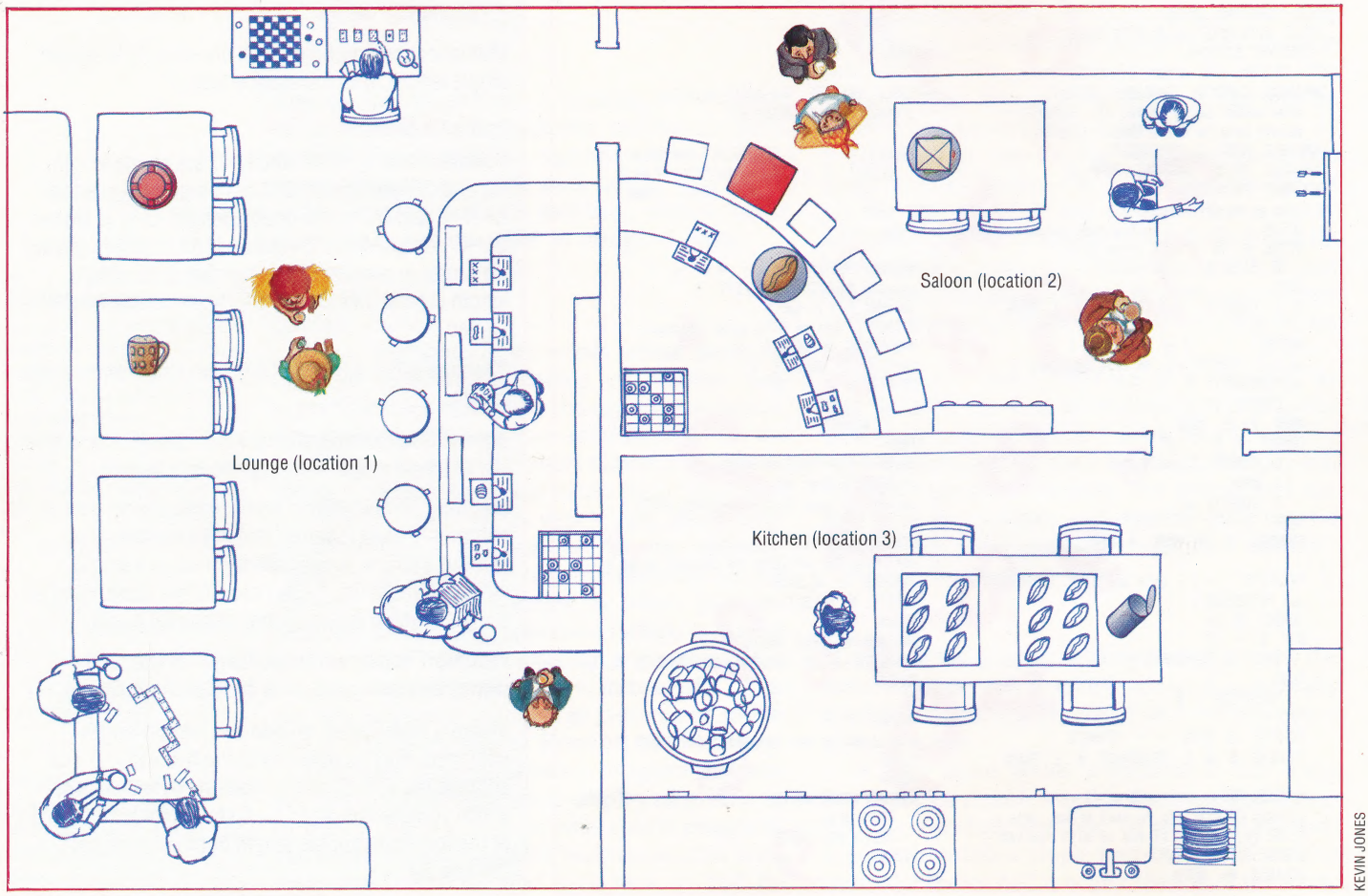
Both these modules will, of course, have further lines added to them in the course of the series as we add other routines and implement the character-handler itself. One benefit of adopting this modular approach is that we'll be able to RUN and test this section of the program as soon as we have typed it in, and then test each successive routine individually as we enter them later.

Note the 'Default values' prompt in line 70. This will enable us, when RUNning the program, to enter our own values for the different character attributes instead of the default values entered in the DATA statements. This 'customisation' facility is very important when designing interactive characters with more than two or three attributes, since it enables you to 'fine-tune' the characters during play. For example, you may find that Sally Short is too moody, or perhaps she moves too quickly from one location to another.

Line 530 is simply a test loop, which will, in turn, call each of the main routines entered so far. We shall be altering this line in future instalments as we add other routines.

The input routine at line 2030 waits for you to press a key. Any key, other than 0, 1, 2, or 3 (ASCII values 48 to 51), is ignored and control returned to the program loop in line 530. Pressing 1, 2, or 3 enters the player into the specified location, where you will be able to see what's going on. You could, of course, devise an input routine that would enable you to address the characters directly, but in our example program all character/player interaction will be carried out by the character-handler, by virtue of our having included the player as an extra character (see page 1492).

Should you decide to adapt the methods we are discussing here for inclusion in one of your own adventure programs, remember that the character-handler can exist independently of the



KEVIN JONES

input routine, as we'll see later when we examine how one might add a simple handler routine to our Haunted Forest program, begun on page 766.

Pressing the 0 key at any time enters the 'Default values' routine at line 2300 and enables you to edit the character attributes as already mentioned and described below.

Lines 2070–2390 provide the three essential display sub-routines. The first, at line 2100, simply prints the description of the current location. When you first run the game, this will be the description of the lounge, location number 1, as set by line 60 in the initialisation module.

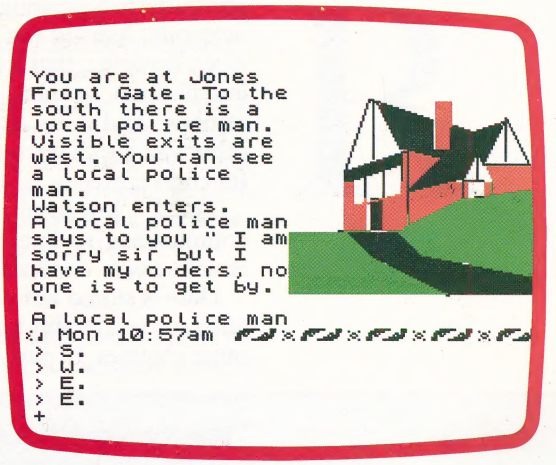
The 'Print visible objects' routine at line 2150 loops through the object array b$ and checks each location entry to see if it corresponds to the player's current location (r). Appropriate object descriptions are then printed to the screen; otherwise, the player is told 'You can see: nothing'. The 'Print visible characters' routine operates in a similar fashion, checking the relevant element of the array c$ and printing a character name if the person in question is present.

The 'Initialise characters' subroutine is called whenever you press 0 during run-time, or if anything other then y or Y is entered in response to the 'Default values' prompt during the initialisation sequence. This routine uses the array d$(11) and the name of each character to print a prompt and asks whether you want to set the values of the character in question. If you answer in the affirmative, the different attributes, whose titles are stored in array d$, are displayed and a value entered for each. Entering a null string for an

Defective Detective

Melbourne House's Sherlock is a good example of a program that provides advanced interactive features wholly in RAM, without loading in additional data from disk or cassette. The player can converse with characters using 'Say to...' and 'Tell me about...', and must solve a number of crimes.

Early versions of the program included some amusing bugs in the character handling, including one that often resulted in Doctor Watson sitting in the same chair at the same time as Holmes


You are at Jones Front Gate. To the south there is a local police man. Visible exits are west. You can see a local police man.
Watson enters.
A local police man says to you " I am sorry sir but I have my orders, no one is to get by. "
A local police man
Mon 10:57am
> S.
> W.
> E.
> E.
+

Initialisation Module

This module will obviously be added to as we introduce new variables and arrays

```
10 REM ¥welcome to the Dog and Bucket¥
20 REM
30 REM .........initialise.............
40 REM
45 GOSUB 4030: REM clear the screen
50 DIM l$(3,5),b$(12,4),c$(7,11),d$(11)
60 r=1
70  PRINT "Default values (y/n)?": GOSUB
 4110: IF i$="y" OR i$="Y" GOTO 90
80  GOSUB 2300: GOTO 100
90  FOR n=1 TO 7: READ c$(n,1): FOR d=2
TO 11: READ c$(n,d): NEXT d: NEXT n
100 FOR n=1 TO 3: READ l$(n,1): FOR e=2
TO 5: READ l$(n,e): NEXT e: NEXT n
110 FOR n=1 TO 12: READ b$(n,1): FOR d=2
 TO 4: READ b$(n,d): NEXT d: NEXT n
120 FOR n=2 TO 11: READ d$(n): NEXT n
```

Program Loop

We will be altering this line in future instalments to call new routines as we add them to our program

```
500 REM
510 REM test program loop
520 REM
530 GOSUB 2100: GOSUB 2150: GOSUB 2240:
GOSUB 2030: PRINT: PRINT: GOTO 530
```

Low Level Subroutines Module

Check the Basic Flavours box for your own particular micro — the version printed is for the Amstrad 464/664 and compatible BASICs — others may differ

```
2000 REM
2010 REM input routine
2020 REM
2030 GOSUB 4110
2040 IF (ASC(i$)<48) OR (ASC(i$)>51) THE
N RETURN
2050 IF i$="0" THEN GOSUB 2320: RETURN
2060 r=VAL(i$): c$(7,2)=i$: RETURN
2070 REM
2080 REM location print
2090 REM
2100 PRINT l$(r,1)
2110 RETURN
2120 REM
2130 REM print visible objects
2140 REM
2150 PRINT "You can see: ";
2160 p=0: FOR b=1 TO 12: IF VAL(b$(b,2))
<>r GOTO 2190
2170 p=p+1: IF p>1 THEN PRINT ", ";
2180 PRINT b$(b,1);
2190 NEXT b
2200 IF p=0 THEN PRINT "nothing";
2210 PRINT: RETURN
2220 REM
2230 REM print visible characters
2240 REM
2250 p=0: FOR c=1 TO 6: IF VAL(c$(c,2))<
>r GOTO 2290
2260 p=p+1: IF p=1 THEN PRINT "You are i
n the company of: ";: GOTO 2280
2270 PRINT ", ";
2280 PRINT c$(c,1);
2290 NEXT c
2300 IF p=0 THEN PRINT "There's no-one h
ere."
2310 RETURN
2320 REM
2330 REM initialise chars subroutine
2340 REM
2350 PRINT: RESTORE: FOR c=1 TO 6: READ
c$(c,1): GOSUB 4070: PRINT c$(c,1);" - "
;: PRINT "Set this char?": GOSUB 4110
2360 IF (i$<>"y") AND (i$<>"Y") THEN FOR
 n=2 TO 11: READ n$: NEXT n: GOTO 2390
2370 FOR d=2 TO 11: READ s$: PRINT d$(d)
;: INPUT i$: IF i$<>"" THEN c$(c,d)=i$
2380 NEXT d
2390 NEXT c: RETURN
```

attribute (just pressing Enter) will skip on to the next one, leaving the original value unchanged. Note, however, that if this routine is called during initialisation, a value must be entered for each attribute and every character must be initialised, otherwise the character-handler will not be able to function correctly.

The 'low-level subroutines' module commencing at line 4000 is used primarily to enable easier conversion for different machines.

Data is stored last of all. Naturally, we shall be adding considerably to this module as we program other routines. In particular, we shall need to store the different messages for the characters.

Once you've entered the listing, RUN it and experiment by moving from location to location pressing the 1, 2, and 3 keys. You can also check the operation of the character editor by pressing 0 and changing the entries for the character locations, moving them from room to room and checking that their new positions are displayed correctly.

We now have an environmental 'shell' into which we can fit our characters and within which we can see them at work. In the next instalment, we shall see how they are to be brought to life.

Main Subroutines Module

To start with, we have only four main subroutines — input, print locations, characters, and objects — and the 'character attributes editor' subroutine

```
4000 REM
4010 REM low level system subroutines
4020 REM
4030 REM clear the screen
4040 REM
4050 CLS: RETURN
4060 REM
4070 REM beep
4080 REM
4090 PRINT CHR$(7);: RETURN
4100 REM
4110 REM get a character from the keyboa
rd
4120 REM
4130 i$=INKEY$: IF i$="" GOTO 4130
4140 RETURN
```

Data Module

Note that the location descriptions have been formatted for a 40-column display. You may have to change these to suit your own micro. You could of course alter any of the default values in lines 6030-6040, in particular the values entered for 'you'

```
6000 REM
6010 REM character data
6020 REM
6030 DATA "Toby Belcher","2","7","10","1
0","7","m","0","0","7","4","Fiona Frappe
","1","8","30","10","8","f","0","0","3",
"5","Steve Swig","1","9","8","10","9","m
","0","0","4","6","Sally Short","2","0",
"20","10","10","f","0","0","5","5"
6040 DATA "Rupert Beer","2","11","10","6
","11","m","0","0","4","6","Molly Mixer"
,"1","12","15","6","12","f","0","0","5",
"5","you","1","0","255","255","0","m","0
","0","0","0"
6050 REM
6060 REM location data
6070 REM
6080 DATA "You are in the lounge of the
Dog and     Bucket. Several shady charact
ers are     gathered together in a corner
 playing    dominoes.  Behind the counter
, Fred the Barman looks his usual cheery
 self.      Exits lead east.","0","0","2"
,"0"
6090 DATA "Behold the saloon of the Dog
and Bucket,which looks as if it could do
 with       extensive redecoration. The f
loor        appears to have been regularl
y hosed     down with beer slops.  Doors
lead west   and south.","0","3","0","1"
6100 DATA "Ugh! This is the kitchen, whe
re the      famous Dog and Bucket Cornish
 Pasties    are prepared for an ever-hung
ry          clientele. You notice a numbe
r of empty cat-food tins, which is stran
ge because there are no cats.","2","0","
0","0"
6110 REM
6120 REM object data (for b$(12,4))
6130 REM
6140 DATA " a glass of beer","2","n","y"
," an empty tin of catfood","3","n","n",
" a Dog and Bucket cornish pasty","1","y
","n"," a bar-stool","2","n","n"," an as
htray","1","n","n"
6150 DATA " a stale ham sandwich","2","y
","n"," a pint of bitter","0","n","y","
a creme de menthe","0","n","y"," a whisk
y and water","0","n","y"," a neat vodka"
,"2","n","y"," a pint of lager","0","n",
"y"," a gin and ginger ale","0","n","y"
6160 REM
6170 REM character attribute data (for d
$(11))
6180 REM
6190 DATA "Location","Strength","Invento
ry","Mood","Own object","Sex","Last char
 (lch)","Last command code (lcd)","Handl
e frequency","Move frequency"
```

Basic Flavours

Spectrum:

```
50   DIM L$(3,5,255),b$(12,3,25),c$(6,11,15),
     d$(11,25)
2040 IF (CODE(i$)<48) OR (CODE(i$)>51)
     THEN RETURN
4090 BEEP .5,3: RETURN
```

Commodore:

```
4050 PRINT "<shift/CLR>": RETURN
4090 POKE 54296,15: POKE 54276,17: POKE
     54277,64
4095 FOR A=1TO 50: POKE 54273, 36: POKE
     54272,85: NEXT: POKE 54273, 0: POKE
     54272, 0: POKE 54276,0: RETURN
4130 GET i$: IF i$="" THEN 4130
```

BBC:

```
4130 i$=GET$
```

R

RING NETWORK

A *ring network* is a type of network configuration in which all the computers within the system are connected together to form a circle. It is perhaps the simplest form of network in use since it requires no master control node or mechanism to select the direction of the message, as all messages are sent in the same direction. However, for heavily-used ring networks, a clock may be necessary to synchronise input and output messages.

As each message passes through a computer in a ring network, the machine checks to see whether the message is intended for it by examining the destination address. If the message is destined for that computer, it is then transferred to the CPU. However, when the message is not intended for that machine or is intended for other computers as well, a copy of the message is made on the output lines and sent on.

ROBOTICS

This is an area of research and development which utilises the programs and microelectronic techniques of both artificial intelligence and engineering technology to produce machines that perform work in the real world and yet are able to follow a pre-programmed sequence of actions or make decisions, based on information received from input devices.

Thus a true *robotic* device will have sense mechanisms attached to receive information, a processing system to analyse the data and output mechanisms to manipulate objects. Each of these three areas is currently the subject of intensive research throughout the world. However, while the application of the software is properly the field of artificial intelligence, it is the hardware implementation and feedback control which is considered applicable to the field of robotics.

Derived from the Czech word for worker (from a play by Karel Capek—Rossum's Universal Robots) in the 1920s, robots have been favourite subjects for science fiction writers ever since. However, like many similar topics, the truth is perhaps somewhat more prosaic. Robots will have a part in the future but the man/machines visualised by science fiction are still a long way off. (For more information on robotics, see our series on the subject beginning on page 601.)

Slave To Work
Robotics is beginning to fulfil its promise of making a huge impact on society. Although the 'robot-slave' of science fiction, like Robby from the film Forbidden Planet, is not yet with us, we are now accustomed to the fact that robots work on automated production lines, performing repetitive or dangerous tasks

ROM

This is an acronym for read only memory. *ROM* chips are random access semiconductor devices which store programs and other data permanently — even when the power is switched off. They are therefore used to contain a computer's operating system, BASIC interpreter and other programs which will always be required by the computer.

The most common type of ROM — and the most convenient for mass production — is 'mask' ROM. Here, the ROM is constructed rather like a printed circuit board. An outline of the circuit to be placed on the ROM is etched onto the chip and the program is constructed that way. The programs and other data held within a ROM are known as 'firmware'.

The other basic type of ROM is programmable ROM (PROM, see page 1365); with this, programs and other data can be written to the ROM for permanent storage. There are, however, several different types of PROM. Among the most common are the EPROM (erasable by shining ultraviolet light on it), and the EEPROM (electrically erasable PROM).

ROUNDOFF ERROR

When large numbers are being calculated you will often find that the least significant digit has been left off the end of the result. This is because, following a binary calculation, the byte that has been assigned for the answer is not large enough to contain the whole result. This will then be shown when the binary calculation is converted into decimal. Although for most applications the last few binary digits do not matter, occasionally this omission can be significant. For example, when one is performing large quantities of number crunching, statistical analysis — where constant calculation will compound the effect of removing the last few digits — or decision-making that relies on high precision, errors will begin to appear. These are known as *roundoff errors.*

Obviously, some of these errors can easily be avoided. For example, avoiding branch decisions which depend on a variable being precise to more than a certain number of digits. However, roundoff errors caused by performing a series of calculations on numbers which have a large number of significant digits, where errors can accumulate, are less easy to avoid. Unless your micro is capable of storing these large numbers, you will have to face the fact that it is just not capable of performing these tasks using BASIC arithmetic routines.

INDUSTRIAL ACTION

Computers now play a vital role in the manufacture of many products, from their conception and design through to their delivery to the consumer. We begin here a short series in which we will outline the important part played by computers in industry today.

To be effective, the manufacturing process—from product design to distribution of the finished goods—requires juggling numerous different factors; time against people, machines against material, design against delivery and a dozen other considerations. In the early 1980s, only the largest £100,000 computers were powerful enough to be of much use to the manufacturing industries. Gradually this has changed, not only because small computers have become more powerful, but also because companies have begun to gain a greater awareness of the contributions computers can make.

It is possible, though in very few instances, to see a product that has been built with the use of computers in every stage of manufacture — from the design stage, right through to the robot assembly. Most of these advanced plants, reasonably enough, can be found in the computer industry itself. The microelectronics companies and personal computer manufacturers are among those at the forefront of this technology.

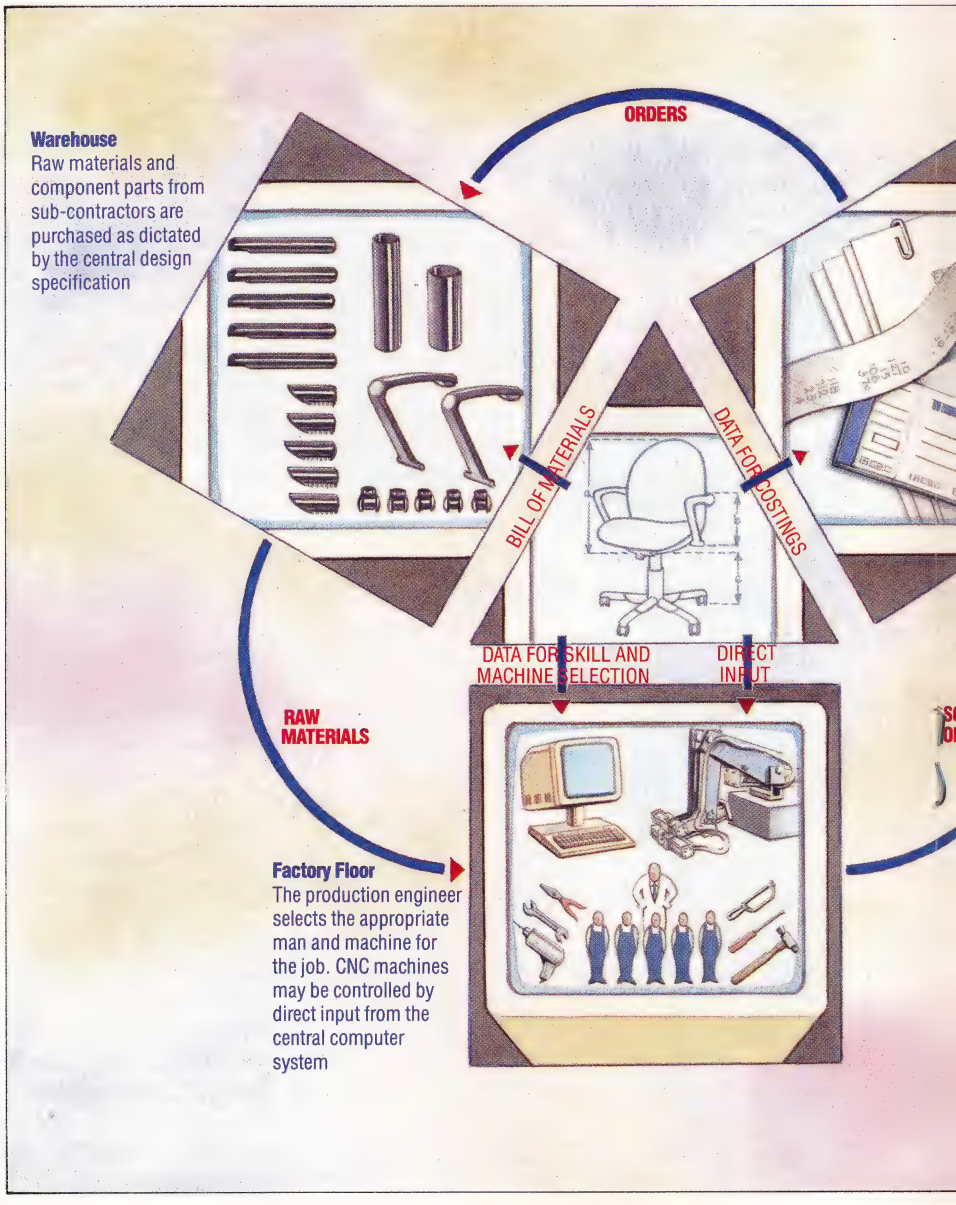
Using computers in every stage of the manufacturing process is known as computer integrated manufacturing (CIM), but names in this application area can be very confusing, as can be seen in our Industrial Jargon box.

Manufacturers work on a co-operative basis, with each department responsible for a separate part of the process. The design and drafting department develops an idea and draws up the technical specifications that will be used as the basis for the entire process. The tools to shape the product will have to be ordered (and possibly made) and the machines will need to be programmed to produce it. The warehouse ensures that stocks of necessary materials are constantly available.

Then there is the financial side of the operation. Before ordering a product, customers want to know what the price will be and the manufacturer has to supply a precise quote. If it is too high, the customer will look elsewhere and if it is too low, the company will lose money. The order must be taken, a delivery date specified and the product must be delivered punctually and paid for at the end of the process. The foreman of the plant has to concern himself with deployment of staff — how many are needed, what skills they possess and when they will be required — as well as ensure that all the orders that have been taken will be delivered at the specified times.

DATA SHARING

However different these tasks may appear, they are all based around what is essentially the same data. This is the key to the CIM approach, in which the computer holds the database that provides the core of information, and the different departments use it as they see fit. The warehouse can look at the description of the product and see what it is made of, the foreman can see what machines are


ORDERS
Warehouse
Raw materials and component parts from sub-contractors are purchased as dictated by the central design specification
BILL OF MATERIALS
DATA FOR COSTINGS
DATA FOR SKILL AND MACHINE SELECTION
DIRECT INPUT
RAW MATERIALS
Factory Floor
The production engineer selects the appropriate man and machine for the job. CNC machines may be controlled by direct input from the central computer system

Aids To Industry
Computers can bring a degree of integration to the manufacturing process previously unattainable. The CAD concept is central to the organisation of the computer-oriented factory, from the design stage through prototyping to final manufacture

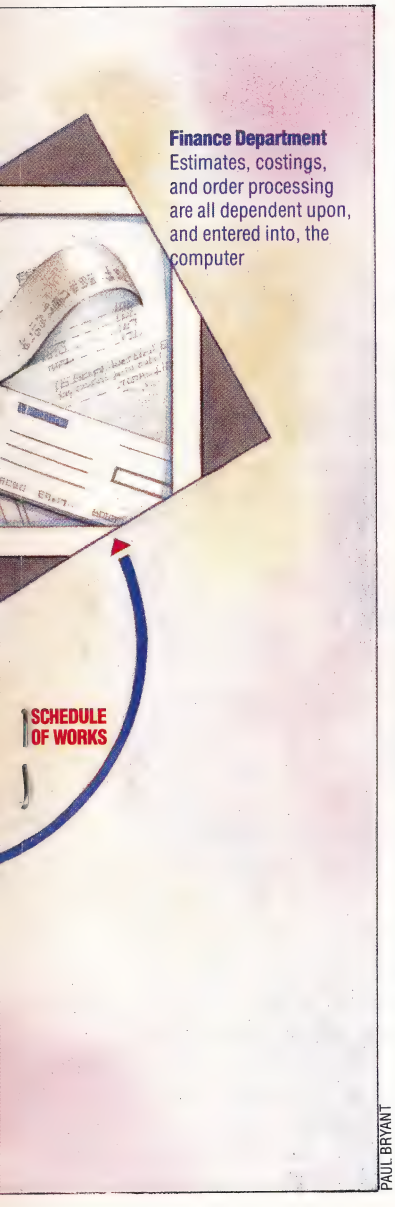
required to build it and the financial departments can see what it costs to produce — all information is included in the database.

Let's now take a look at the data involved and the different steps involved in processing it. The crucial data is, of course, the *description of the product*, which is produced by the design department. Computer aided design (CAD; see page 235) systems have helped in this process for many years now. Few products are comprised of just a single element but, rather, are made up of parts within parts. If we look at a car, for example, we can clearly see that there is an overall design, but it is made up of thousands of different components — from the wiring in the rear lights to the timing belt in the engine.

The best CAD systems can build up an image of the product part by part, placing them layer upon layer until the entire image is complete, in a three-dimensional representation. From this image, the central core of information, simulation and analysis programs can check not only the design, but how well the product will work in practice.

The bill of materials (BOM) is central to the successful operation of the warehouse, providing a complete list of all the parts in the product, which can be broken down to a list of the raw materials that are needed. The warehouse's interest is in matching the list of components against what is actually in stock. The department will want to know if all the parts are there and if re-ordering will be necessary after production begins. If so, they will have to inform the purchasing department of both the relevant supplies needed and from whom they usually receive their goods.

The designs of the product inform the production engineers, who are responsible for the manufacture itself, what demands will be made on the machines and the staff. From this point, it is a small step to measuring the requirements against the equipment and staff available and drawing up a schedule of work.


Finance Department
Estimates, costings, and order processing are all dependent upon, and entered into, the computer
SCHEDULE OF WORKS
PAUL BRYANT

THE CIM DATABASE

The CIM is used to plan not only the manufacture of the single product, but the scheduling of several different products all going through various stages of development on the same factory floor. The successful system will avoid situations in which, for example, bolts have been made but are useless because none of the nuts are ready.

In other words, then, the database concerning a product, which was compiled at the time of drawing, can detail all the processes that go into making it. Matching this information against an existing schedule of work produces a timetable, and the same database will tell the accounts office about the production costs. It will measure the quantities of materials to be used and by scheduling machine time and staff skills, will give the information needed to provide the customer with a realistic quote.

The data of the precise picture held in the drawing, which was produced by the design department, can then be passed straight to the computer-controlled machines that will perform the operation. There is now a web of information based on the design office's original ideas. The warehouse manager, tool department, sales office, accounts clerks, foremen and even the despatch office can all take advantage of the same pool of information about the product.

In some of the newer industries computers have become so much a part of the manufacturing process that they couldn't reasonably continue without them. The design, for instance, of today's microprocessors is now too complicated to be carried out without computers. The process would take too long if done by hand and designs drawn up by draughtsmen would be impossible to test without actually building the chip. That in itself would be too expensive to be feasible.

CIM is the central management of information about manufacturing processes, and to be effective it needs to be fed with reliable information. Various computing techniques have been developed to this end and the following instalments will look at some of these and the variety of ways in which they help the manufacturing industry.

Industrial Jargon

Computer-based applications in industry are shrouded in a jargon of acronyms. Here are a few of the more commonly used terms:

AMT Advanced Manufacturing Techniques. This is a general term referring to CAD, CAM and CAE

CAD Computer Aided Design. This is sometimes used to refer to Computer Aided Drafting, a less sophisticated application, which is capable of drawing pictures, but not analysing them

CAM Computer Aided Manufacture. Refers to any area in which computers are used in the manufacturing process

CAE Computer Aided Engineering. The use of data (usually the CAD database) to carry out an analysis of a design

NC machine Numerical Control machine. Used for shaping parts in the manufacturing process

CNC machine Computer Numerical Control machine. Like computers themselves, early NC machines used to be driven by paper tape instructions. Now computers can be used to program the machines directly

CIM Computer Integrated Management. Covers the entire process of automating factory management

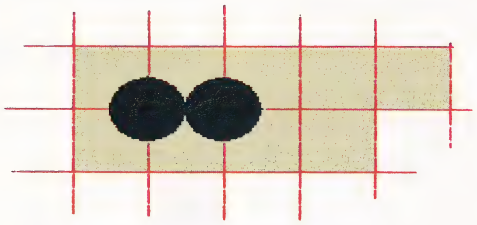
BOM Bill Of Materials. The list of components required for a product to be manufactured

GROUP AWARENESS

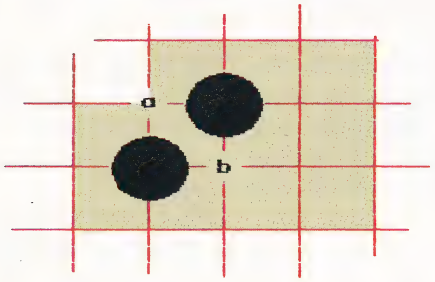
As our Go programming project draws to a conclusion, we look at perhaps the most important consideration when playing the game—connecting groups of stones. By incorporating the various connection operations and pattern-matching routines, we can ensure that the program will deal with all eventualities.

At the end of the game, territory is only officially surrounded if it is enclosed by a continuous line of stones. This will consist of unbroken links, as in figure 1, or by diagonal links (figure 2), where, if White plays in 'a', Black can form two unbroken links by playing 'b', and vice versa. However, it is usually only towards the end of the game that these solid links are formed.

1

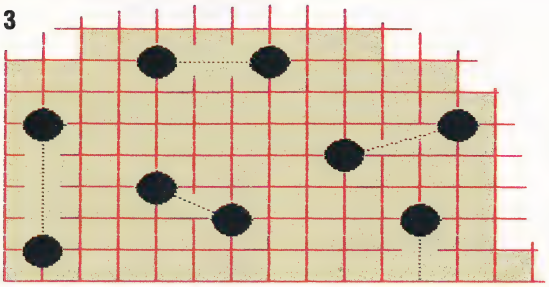


2



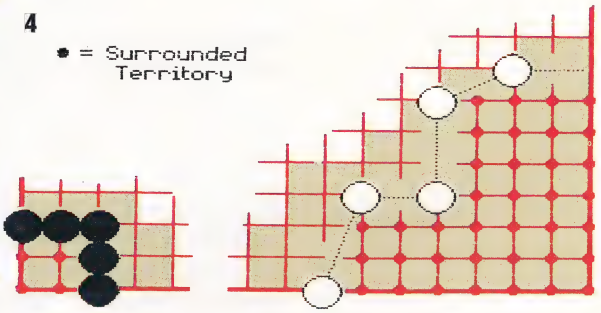
For most of the game, stones will be separated by one or more spaces, forming imaginary links, either with other stones, or with the edge of the board (figure 3). In these cases, solid links will only be formed when there is a possibility of these 'connections' being breached by the opponent. It is not absolutely guaranteed that these stones will eventually be connected, but it is quite likely.

3



Obviously, if we want our Go program to play a reasonable game, we must tell it something about these vague connections. If we only teach the program that solid connections of adjacent stones can be used to surround territory, the program is going to be in trouble. For example, in figure 4, while our program (Black) is surrounding four points of territory, the opponent has used the same number of stones to surround a likely territory of 37 points!

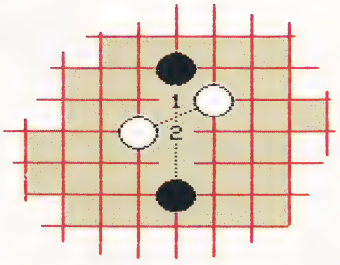
4



Having decided that we want the computer to understand connections, we have to decide how to program it. There are four major connection operations that we want the computer to handle. These are:

- Defending a connection between two stone groups when the opponent moves into a possible attacking position.
- Attacking an opponent's connection by placing a stone between the two tenuously connected stones.
- Playing a start_attack stone. This will either be preliminary to attacking by placing a central stone, or will provide a back-up for a centrally placed stone.
- Starting a connection by placing a stone at a distance from a previously placed stone. By using the weighting system—developed in the last instalment (see page 1495)—when scoring this stone, a reasonably good position will automatically be found, hopefully surrounding territory.

5

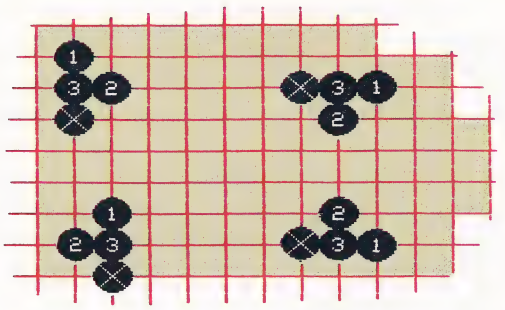


These operations have been programmed using simple pattern-matching techniques. Look at a typical situation as shown in figure 5. White ideally wants to break the connection between the two black stones by playing in position '2', and Black wants to defend by playing in the same position. Simultaneously, Black wants to break the white stone connection at either '1' or '2', while White wants to defend by playing on these points. In this particular case, White has the advantage, having the more closely connected stones.

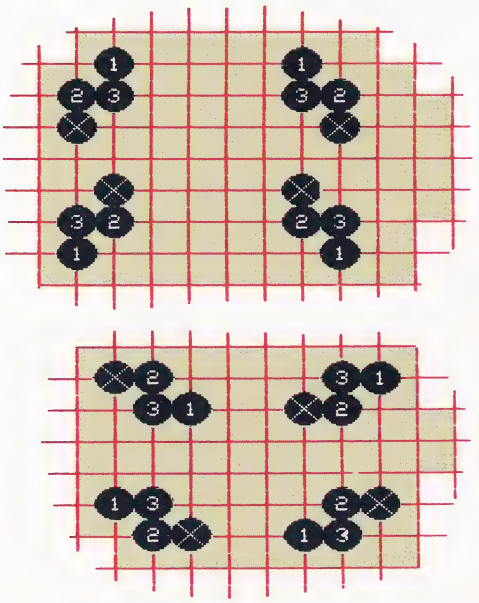
It's possible to implement all four connection operation types using a single set of patterns.

These patterns are set up by the routine beginning at line 790, which defines the patterns shown in figures 6, 7, 8 and 9. In all cases, the crossed stone is our current stone (point zero). The DATA statements then give offsets from this point on the board% board map. So, +16, −16, +1 and −1 correspond to north, south, east and west, respectively.

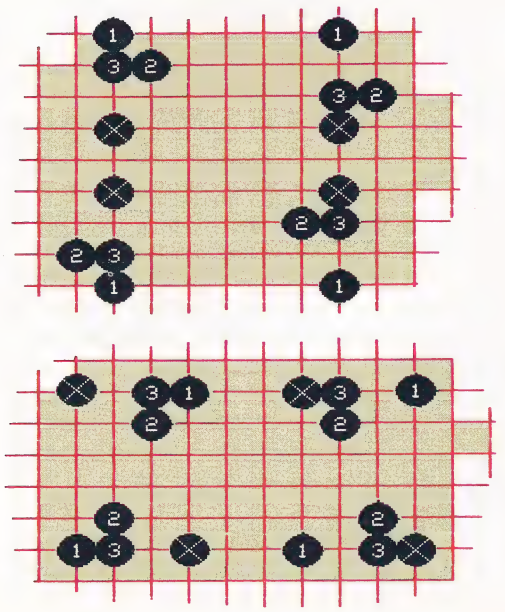
6



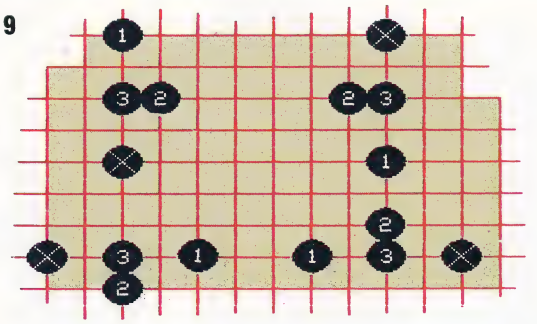
7



8



9



Since every stone will eventually be considered as the current stone, it isn't necessary to implement all orientations of each pattern. For example, in figure 9 we don't have a pattern with the current stone at the bottom (0), the first stone at the top (+64), and stone 2 to the left of stone 3 (+31 and +32). However, later in the board search, the stone at present marked 1 will be checked as the current stone. The pattern shown at the top right of figure 9 will then handle the pattern orientation just described. Thus symmetrical patterns only require four orientations in memory, and non-symmetrical patterns require eight.

You'll notice that every other DATA statement is just the negation of the previous line, so we can reduce the number of patterns still further, if we were short of memory space, by specifically trying the negation of all the held patterns.

The pattern-matching routines work by calculating the board positions of these offsets, and thus finding the colour of the stones, if any, in the relative positions. A set of rules is then applied to decide whether or not to play a stone. If a stone is to be played, then the position is assigned a score based on our board-weightings; at the end of the routine, the stone with the highest score is used.

The routines are checked in order, so if a move is found by, say, the attack_connection routine, then the start_attack and start_connection routines will not be checked.

Assuming Black is to play, the rules used to decide whether or not to play a stone are:

defend_connection:

IF	crossed stone	= BLACK
AND	stone one	= BLACK
AND	stone two	= WHITE
THEN	try move at stone three.	

attack_connection:

IF	crossed stone	= WHITE
AND	stone one	= WHITE
AND	stone two	= BLACK
THEN	try move at stone three.	

start_attack:

IF	crossed stone	= WHITE
AND	stone one	= WHITE
AND	stone three	= EMPTY
THEN	try move at stone two.	

start_connection:

IF	crossed stone	= BLACK
AND	stone three	= EMPTY
THEN	try move at stone one.	

The four connection routines shown here are called from lines 2580 to 2610 of the black_move routine.

This will now handle connections between stones on the board. However, if you look back to figures 3 and 4, you'll see that we also have connections between stones and the edge of the board, which are just as important. The PROCboundary routine is designed to deal with this. You may remember that our board is actually defined DIM board% 256 for a 15-by-15 board. This gives us our wrap-around over 256 bytes, and also leaves us with a single-stone border surrounding the board. PROCboundary fills this border with the value passed as a parameter to V%. By ensuring that the pattern-matching routines search the whole board, including the border (from 1 to 255), connections with the edge of the board will also be checked. Always remember to reset the border to zero when exiting the routine by calling PROCboundary with a value of zero.

Due to the speed of the program the number of patterns has been kept to a minimum. However, feel free to add or change the data starting at line 860. If you do, don't forget to change the loop parameters in lines 2940, 3110, 3260 and 3410.

Module Five

BBC Micro:

```
  280 PROCread_patterns
  780 :
  790 DEF PROCread_patterns
  800 LOCAL L%
  810 DIM pat% 71
  820 RESTORE 860
  830 FOR L% = 0 TO 71
  840   READ pat%?L%
  850   NEXT
  860 DATA 32,17,16,2,-15,1
  870 DATA -32,-17,-16,-2,15,-1
  880 DATA 33,16,17,31,16,15
  890 DATA -33,-16,-17,-31,-16,-15
  900 DATA -14,1,-15,18,1,17
  910 DATA 14,-1,15,-18,-1,-17
  920 DATA 48,33,32,48,17,16
  930 DATA -48,-33,-32,-48,-17,-16
  940 DATA 3,-14,2,3,-15,1
  950 DATA -3,14,-2,-3,15,-1
  960 DATA 64,33,32,4,-14,2
  970 DATA -64,-33,-32,-4,14,-2
  980 ENDPROC
  990 :
 1000 REM*******************************
 2580 IF location%=0 THEN PROCdefend_con
nection:T$="DEF"
 2590 IF location%=0 THEN PROCattack_con
nection:T$="ATT"
 2600 IF location%=0 THEN PROCstart_atta
ck:T$="SAT"
 2610 IF location%=0 THEN PROCstart_conn
ection:T$="SCN"
 2890 :
 2900 DEF PROCdefend_connection
 2910 LOCAL A%,B%,C%,D%,P%,S%,hi,score
 2920 PROCboundary(black%)
 2930 FOR A%=1 TO 255 : S%=board%?A% : I
F S%<>black% THEN 3010
 2940   FOR P%=pat% TO pat%+70 STEP 3
 2950     B%=board%?((A%+?P%) AND 255):C
%=board%?((A%+?(P%+1)) AND 255)
 2960     IF B%<>black% OR C%<>white% TH
EN 3000
 2970     D%=(A%+?(P%+2)) AND 255:score=
RND(1)+weight%?D%
 2980     IF (D% AND 240)=0 OR (D% AND 1
5)=0 OR score<=hi THEN 3000
 2990     IF FNlegality(D%,black%)=0 AND
 clib%>2 THEN hi=score:location%=D%
 3000     NEXT
 3010   NEXT
 3020 PROCboundary(0)
 3030 ENDPROC
 3040 :
 3050 REM*******************************
 3060 :
 3070 DEF PROCattack_connection
 3080 LOCAL A%,B%,C%,D%,P%,S%,hi,score
 3090 PROCboundary(white%)
 3100 FOR A%=1 TO 255 : S%=board%?A% : I
F S%<>white% THEN 3180
 3110   FOR P%=pat% TO pat%+70 STEP 3
 3120     B%=board%?((A%+?P%) AND 255):C
%=board%?((A%+?(P%+1)) AND 255)
 3130     IF B%<>white% OR C%<>black% TH
EN 3170
 3140     D%=(A%+?(P%+2)) AND 255:score=
RND(1)+weight%?D%
 3150     IF (D% AND 240)=0 OR (D% AND 1
5)=0 OR score<=hi THEN 3170
 3160     IF FNlegality(D%,black%)=0 AND
 clib%>2 THEN hi=score:location%=D%
 3170     NEXT
 3180   NEXT:PROCboundary(0)
 3190 ENDPROC
 3200 :
 3210 REM*******************************
 3220 :
 3230 DEF PROCstart_attack
 3240 LOCAL A%,B%,C%,D%,P%,S%,hi,score
 3250 FOR A%=1 TO 255 : S%=board%?A% : I
F S%<>white% THEN 3330
 3260   FOR P%=pat% TO pat%+70 STEP 3
 3270     B%=board%?((A%+?P%) AND 255):D
%=board%?((A%+?(P%+2)) AND 255)
 3280     IF B%<>white% OR D%<>0 THEN 33
20
 3290     C%=(A%+?(P%+1)) AND 255:score=
RND(1)+weight%?C%
 3300     IF (C% AND 240)=0 OR (C% AND 1
5)=0 OR score<=hi THEN 3320
 3310     IF FNlegality(C%,black%)=0 AND
 clib%>2 THEN hi=score:location%=C%
 3320     NEXT
 3330   NEXT
 3340 ENDPROC
 3350 :
 3360 REM*******************************
 3370 :
 3380 DEF PROCstart_connection
 3390 LOCAL A%,B%,C%,P%,S%,hi,score
 3400 FOR A%=1 TO 255 : S%=board%?A% : I
F S%<>black% THEN 3470
 3410   FOR P%=pat% TO pat%+70 STEP 3
 3420     C%=board%?((A%+?(P%+2)) AND 25
5):IF C%>0 THEN 3460
 3430     B%=(A%+?P%) AND 255:score=RND(
1)+weight%?B%
 3440     IF (B% AND 240)=0 OR (B% AND 1
5)=0 OR score<=hi THEN 3460
 3450     IF FNlegality(B%,black%)=0 AND
 clib%>2 THEN hi=score:location%=B%
 3460     NEXT
 3470   NEXT
 3480 ENDPROC
 3490 :
 3500 REM*******************************
 4410 :
 4420 DEF PROCboundary(V%)
 4430 LOCAL X%,Y%
 4440 FOR X%=0 TO 15
 4450   Y%=16*X%
 4460   board%?X%=V%
 4470   board%?Y%=V%
 4480   NEXT
 4490 ENDPROC
 4500 :
 4510 REM*******************************
 4520 REM******END OF PROGRAM**********
```

RELAX, DO IT

Frankie Goes To Hollywood, from Denton Designs, is based neither on the band's music nor on the members themselves. It is, however, a cleverly constructed program that incorporates the band's philosophy of escaping from 'Mundanesville' to the 'Pleasure Dome'.

Frankie Goes Searching
The object of Frankie Goes To Hollywood is to search a number of rooms for objects. When a drawer is opened, the contents will be displayed in a window on the screen. It is also possible to 'enter' certain pictures and play an arcade-style game

LIZ HEANEY

Frankie Goes To Hollywood is based not so much on the group's music, but rather on their philosophy of escaping from the everyday world — 'Mundanesville' — to the 'Pleasure Dome', and becoming a 100% 'real person' on the way.

The game is a product of Denton Designs, a programming team from Liverpool — where the band also come from. Originally employed by Imagine, which went into liquidation in 1984, the team now work on a freelance basis for a number of software companies, producing, among other things, the highly rated Shadowfire (see page 1321).

Having a number of similarities to its predecessor, Frankie is an adventure game with a number of places to explore (in this case a row of terraced houses). While moving around the houses, it is possible to search the furniture for objects that may be useful to you. These objects are hidden in drawers and unlikely places like washing-machines.

Most adventures require that you type in commands, however Denton Designs have adopted the method of exploring objects that was first used on Shadowfire. Once you have found an object, it can be picked up and dropped by placing the cursor over it and pressing the fire button. Among the most common objects you will discover are four small icons that correspond to the four components of your personality. These components are shown on the right-hand side of the screen and increase as you progress towards becoming a 'real person'. The icons that correspond to these components are 'pleasure pills', and taking them increases your rating in that area of you personality.

You may only carry eight objects at any one time and as your inventory of objects increases, the temptation is to take the pleasure pills as soon as you find them. However, they are more useful at a later stage of the game and so it is worthwhile holding on to them for a while.

Another set of objects that continually appear are videos that allow you to enter a completely different facet of the game. Many of the houses have television sets and it is possible to insert the video cassettes into them. Apart from gaining a number of points towards the score, the video will appear in a window on the screen. By moving your 'person' into the video screen, the program changes into one of a number of arcade-style games. It is during these games that eating the pleasure pills will boost your score to its maximum. In some of the houses, you will find pictures hung on the wall and it is also possible to enter these and play an additional arcade game.

There are also several floppy disks to be found in the houses. At first, there seems no use for them as there appears to be no computer in which to place them. However, hidden somewhere in the game is a computer room and it is up to you to find it. Naturally, when you finally get to this room, the information on the disk will assist you in your quest to becoming a real person.

In the course of your travels you will discover that a murder has been committed and it is one of your tasks to identify the murderer. Clues to the murder are flashed up on the screen at intervals. For example, one clue might be The Killer is a socialist, whereas another could be Miss Mundane always votes Tory, thus eliminating Miss Mundane from the list of suspects.

Although many of the features of Frankie Goes To Hollywood have been borrowed from other games, it is definitely good enough to stand in its own right. The addition of so many facets in a single game is certainly unusual and should entertain games players for a considerable time to come.

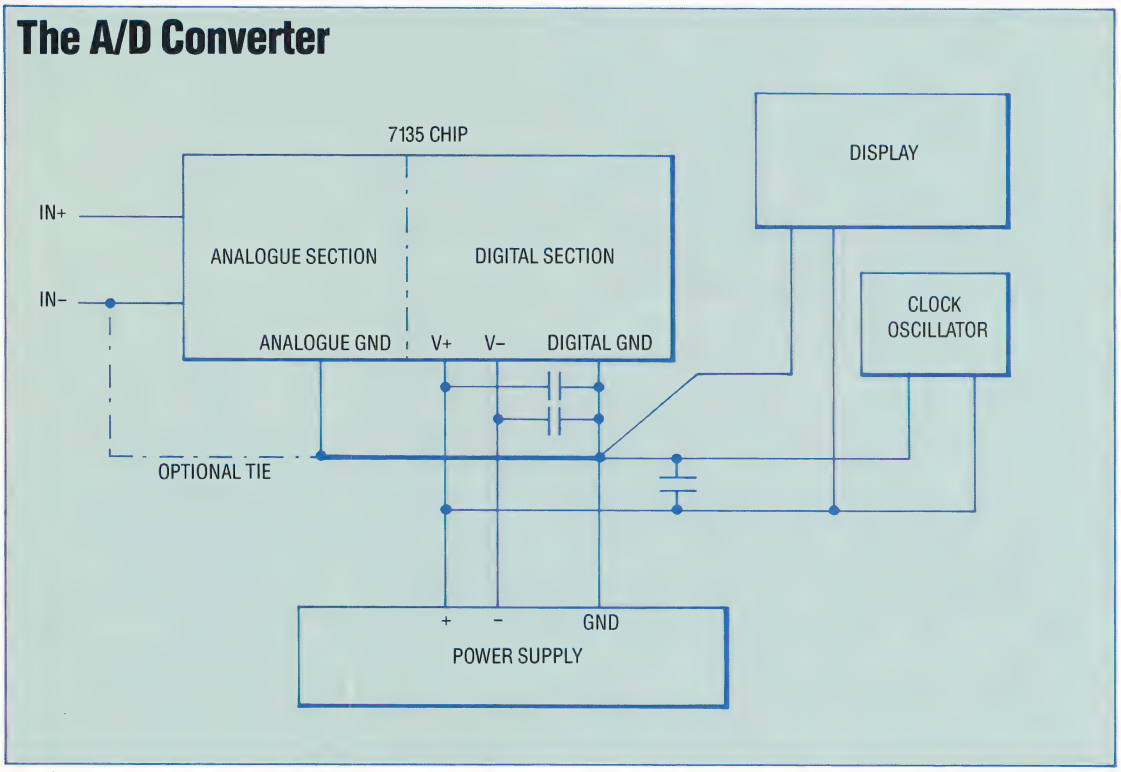
Frankie Goes To Hollywood: For the Sinclair Spectrum and Commodore 64
Price: £9.95 cassette, Commodore 64 disk £12.95
Publishers: Ocean Software, Ocean House, 6 Central Street, Manchester
Authors: Denton Designs
Format: Cassette or disk
Joysticks: Required

METERING REQUIREMENTS

Before constructing our DVM, it is worth examining the 7135 A/D converter chip (the heart of the DVM) in detail. We look at the individual pins of the chip and discuss some important 'dos and don'ts' regarding the board layout and component selection.

Power Principles
It is very important that in considering our DVM circuit layout we ensure that the analogue and digital sections of the design do not come into contact with each other. The general wiring scheme shown here minimises ground loops and the accidental coupling of digital signals into the analogue circuit

The A/D Converter



KEVIN JONES

By first referring to the circuit diagram on page 1464, which shows the pin-out of the 7135 and the names assigned to the pins, we will work our way through them in the order of the pin numbers. A full parts list will be given later.

1. V−: the 7135 requires both a positive and a negative power supply. The 'absolute maximum ratings' of these inputs is +6v for the positive rail and −9v for the negative. To simplify the power supply requirements, we will use +/− 5v. The −5v supply current is typically 0.8mA, so a very small and simple power supply is all that is reqiured. In our design, the unit will be powered from the mains, but a 6v battery could also be used (both TTL and CMOS chips will work happily from a 6v supply). Details of the mains power supply will be given during the constructional phase of the series.

2. Reference: The reference voltage for the 7135 (or any dual-slope A/D converter) is critical to its overall accuracy. The full scale deflection (fsd)—the maximum output value—of a dual-slope A/D converter is exactly twice the reference voltage, so, for a meter with an fsd of 2v, the reference voltage should be exactly 1.000v.

There are any number of ways of deriving an accurate 1v reference, but we recommend using a type 9491 1.22v precision band gap reference diode. The required 1v reference is then derived by using a potentiometer to bring down the voltage to the level required. Do not be tempted to use a cheap volume control-type pot. Use a top-quality multi-turn (ten-turn or more) trim-pot. Borrow a good DVM and adjust the trim-pot for 1.000v. Better still, rig up a potential divider across the output of low-voltage bench power supply and adjust it for a reading on the borrowed DVM of 1.9999v, feed this into the input of your DVM and adjust the reference trim-pot for a reading of 1.9999v on your display.

If you do not have access to an accurate DVM, the next best approach is to create a reasonably accurate voltage source from two 1.35v mercury cells in series and a potential divider constructed from a 36K-ohm and a 100K-ohm resistor. This will give approximately 2v to use as a test input to the DVM. Again, adjust the reference trim-pot for a reading on the display of 1.9999v. Use close-tolerance resistors (one per cent or better) and new mercury cells.

3. Analogue Ground: The innocent-looking pin on the 7135 is more likely to be the cause of trouble with your DVM than any other. The problem comes from the fact that there are two separate ground signals—analogue ground and digital ground (digital ground is on pin 24). The analogue signals, + and − power supplies and ground circuits of the analogue and digital sections should

be kept strictly separated. What is required is a wiring scheme that minimises the possiblity of digital signals getting into the analogue section and vice versa, though the latter instance is not so important.

Circuit designers would tell you that you should adopt good layout principles, but this is not very helpful if you don't know what good layout principles are. We won't go into the reasons behind the layout requirements too deeply, and will confine ourselves to simply telling you what you should and should not do.

All the analogue ground points (the junction of the reference diode and the bottom end of the trim-pot connected to pin 3) and the earth end of the resistor connected to the diode (which is connected to the integrating and auto-zero capacitors on pins 4 and 5) should be soldered to a single point. Solder the anode of the reference diode and the 100K-ohm resistor (connected to Int. Out. via a diode) to a single point.

PRECAUTIONARY MEASURES

It is equally important to connect all the digital ground points to a single point. Notice that the circuit diagram (see page 1464) uses a different symbol for the two types of ground—⏚ for analogue grounds and ⏚ for digital grounds. Having brought all analogue ground connections to a single point, and all digital grounds to another, separate, single point, the two grounds must be tied together. This should be done by using a single length of heavy-gauge solid copper wire.

Failure to observe these precautions will lead to a myriad of complications. Readings on the display will be inaccurate, the display may cycle through several alternative (wrong) readings, the DVM may miss readings, and the whole thing may oscillate or seize up.

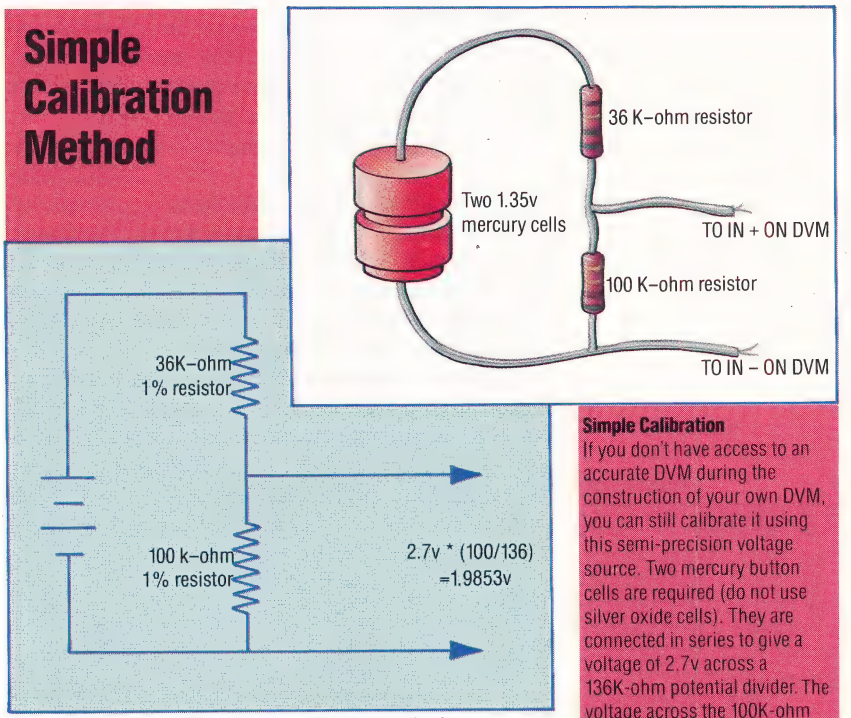
The power supply to the clock, the display driver and to the display itself should be adequately de-coupled using both high-value electrolytic capacitors and low-value disc ceramic capacitors. Just as importantly, the analogue parts of the circuit (such as the reference voltage diode, the integrating resistors, capacitors and so on) should be on the far side of the circuit board from the digital parts of the circuit (such as the CLOCK signal, the STROBE and the BUSY lines).

If this is not done, signals such as BUSY and STROBE, that change from one logic level to another during the course of a reading, are likely to get fed back into the input and cause countless errors. The rule is: keep the analogue and digital components of the circuit physically separate on the board, and pay attention to the layout of the ground lines so that ground loops don't create further problems. The Power Principles box illustrates the general way in which the power supply should be arranged.

Thirdly, any unused digital signals (such as STROBE, UNDER-RANGE, RUN/HOLD and so on) should not be left floating; they should either be pulled up using 4.7K-ohm resistors or pulled down using 440ohm resistors.

4. Int. Out. (Integrater Output): Integrating-type A/D converters such as the 7135 depend on the assumption that the change of voltage across the integrating capacitor is exactly proportional to the current flowing into it. The absolute value of this capacitor is not very critical—0.4 or 0.47µF (microfarads) is fine—but very low dielectric absorption is vital. Polypropylene capacitors are best, but polystyrene are almost as good and polycarbonate capacitors may also be used. Do not use any kind of electrolytic capacitor.

5. A-Z IN: The capacitor used here can be of any quality. The value is not critical, though the bigger it is, the better. Use polypropylene, polystyrene or polycarbonate capacitors, not electrolytics.



Simple Calibration

If you don't have access to an accurate DVM during the construction of your own DVM, you can still calibrate it using this semi-precision voltage source. Two mercury button cells are required (do not use silver oxide cells). They are connected in series to give a voltage of 2.7v across a 136K-ohm potential divider. The voltage across the 100K-ohm resistor will be 1.9853v ideally, but could range unpredictably to around +/– two per cent of this, provided that one per cent tolerance resistors are used. One per cent metal film resistors in these values are available for a few pence from Maplins. 36K-ohm is one of the resistor values from the 'extended range'. Do not try to build it up from 'standard range' series resistors or you risk losing accuracy. The trim potentiometer should be adjusted until the DVM display reads 1.9999v

6. BUF OUT (Buffer Output): This is the point to which the integrating resistor is connected. The value of the integrating resistor is derived from:

$$R = \frac{\text{full scale voltage}}{20\mu A}$$

Any 100K-ohm resistor will do, though a close-tolerance (one per cent or better) metal film resistor would be better than the sort of low-tolerance carbon-composition resistors one tends to have lying in the bottom of a scrap box.

The diode connected to the integrator output (pin 4) is to correct a small roll-over error. Any low-signal silicon type of the sort that scrap boxes are usually full of will do. Although inexperienced circuit designers usually like to have a specific type recommended, the fact is that almost any diode will do, though it should be silicon rather than germanium, and signal rather than rectifier. Whole bags of such diodes can generally be purchased for a pound or less.

PICTURE PERFECT

We conclude our investigation of useful aspects of the Spectrum's operating system available to the machine code programmer, by examining different methods of manipulating the display. The display and attributes files are discussed, and utility listings provided.

Graphics and screen handling on the Spectrum are not really all that difficult. The method we'll be looking at first is to use the routines in ROM that are used by the Spectrum's OS.

To address the screen, channel S must first be opened by selecting stream 2. For a normal print operation the RST#10 routine is by far the easiest method. But if we wanted to print a whole string of characters, a small routine would have to be written to read in each character one at a time, and pass each one over to the RST#10 routine until all the characters had been printed.

However, rather than writing your own routine, there is already one in the ROM which does the job for you. To call this routine, simply load DE with the address at which the string is stored, load BC with the length of the string and call the Print String routine at #203C. A demonstration of this can be seen in our first listing.

```
                ORG  60000
1172EA          LD   DE,MSG
3E02            LD   A,2                ;select stream 2
CD0116          CALL #1601              ;open channel S
1172EA          LD   DE,MSG             ;point DE to string
012600          LD   BC,38              ;load BC with string len
CD3C20          CALL #203C              ;call PRINT-STRING
C9              RET
160A06    MSG:  DEFB 22,10,6            ;codes for AT 10,5
54484520        DEFM "THE HOME COMPUTER"
160C07          DEFB 22,12,7
41445641        DEFM "ADVANCED COURSE"
```

There is another useful ROM routine — the Print a Floating Point Number routine (PRINT-FF for short) at #2DE3. We have already shown you how to evaluate a numeric expression and place it on the calculator stack using the routine at #1C82 (see page 1498). The PRINT-FP routine can make life a lot easier, since printing a number in machine code is quite a difficult task. To print a number, first place it on to the calculator stack by loading it into either the BC register pair or the A register. The STACK-A routine is called at #2D28 and the STACK-BC routine at #2D2B. Once on the stack, printing is a simple matter of calling the PRINT-FP routine.

```
                ORG  60000
013930          LD   BC,12345           ;get number in BC
CD2B2D          CALL #2D2B              ;BC onto calc stack
3E02            LD   A,2
CD0116          CALL #1601              ;open channel S
CDE32D          CALL #2DE3              ;print no from stack top
C9              RET
```

The real power of the Spectrum's graphics becomes more apparent when we use the high-resolution commands PLOT and DRAW. The first of these has been dealt with in an earlier part of the course (see page 1317). DRAW is just as easy to use as PLOT, except that a few more parameters are needed. We already know that the command attempts to draw the straightest possible line between the last point plotted and a point that is defined relative to the origin. So, if a line were drawn 20 pixels up and 40 pixels to the left, the DRAW command would read DRAW -40, 20.

In order to explain the use of the DRAW command in machine code we will use the command syntax DRAW x,y. When executing a machine code DRAW instruction, the x and y parameters must be considered separately. To simplify matters we will leave out any curve factors for the time being. If a return to BASIC is to be made at some point the value of the alternative H'L' register pair must be preserved, as the DRAW routine will corrupt the data contained in it.

The absolute value of x (ABS x) is passed over in the C register, ABS y is passed over in the B register and the sign values for y and x are passed to the routine in D and E respectively. Values greater than -1, 0 and 1 can be used because the routine uses these values as the actual increment that is added to the last pixel plotted to attain the position of the next pixel in the line.

This means that if a sign value of 2 or 3 were used, the line would be two or three times longer than normal, but would be drawn with a stippled effect similar to that available on the QL. Note that the sign values used by the routine have to be in two's complement notation. Examples of both types of drawing can be seen in the third listing.

```
                ORG  60000
D9              EXX
E5              PUSH HL                 ;save H'L'
D9              EXX
0658            LD   B,88               ;Y coordinate
0E64            LD   C,100              ;X coordinate
CDE522          CALL #22E5              ;PLOT 100,88
0E28            LD   C,40               ;ABS x=40
0614            LD   B,20               ;ABS y=20
1EFF            LD   E,255              ;SGN y=-1
1601            LD   D,1                ;SGN x=1
CDBA24          CALL #24BA              ;DRAW -40,20
3E58            LD   A,88
327D5C          LD   (23677),A          ;X org=88
3E32            LD   A,50
327E5C          LD   (23678),A          ;Y org=50
0E28            LD   C,40
0614            LD   B,20
1602            LD   D,2                ;SGN x=2 - stipple line
1E02            LD   E,2                ;SGN y=2 - stipple line
CDBA24          CALL #24BA
D9              EXX
E1              POP  HL                 ;restore H'L'
D9              EXX
C9              RET
```

The information in the two tables we provide, encapsulates all we have just discussed in this section on graphics and screen handling. The first table lists the most useful routines we have used and the second contains the handiest system variables connected with the routines mentioned in the first table.

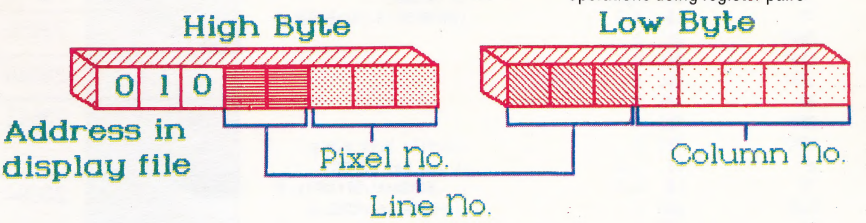
THE DISPLAY FILE

The Spectrum's screen memory is arranged in two parts. The first one is called the *display file* and is used for holding all the information about the actual pixels. The second is the *attributes file*, which stores the colour information for the screen. The display file runs from 16384 through to 22527, comprising 6144 bytes in all. Now type in this program and RUN it:

```
10 FOR F=16384 TO 22527
20 POKE F,255
30 NEXT F
```

When the program runs you will see that the screen slowly fills up. Take a closer look and you will notice that it seems to fill up in thirds. An even closer scrutiny of the screen will allow you to see the actual lines of pixels being drawn. However, contrary to what you might expect, these lines are not drawn consecutively. The first line of pixels appears at the very top of the screen, but the next line does not appear below the first, but eight lines down, the third line sixteen lines down, and so on. When the sixty-fourth line is reached, the process reverts to the top of the screen and the second line is filled in, followed by the ninth. This process continues until the top third of the screen has been filled. The whole business is then repeated on the second part of the screen and, finally, on the last third. This process is explained diagramatically in the figure.

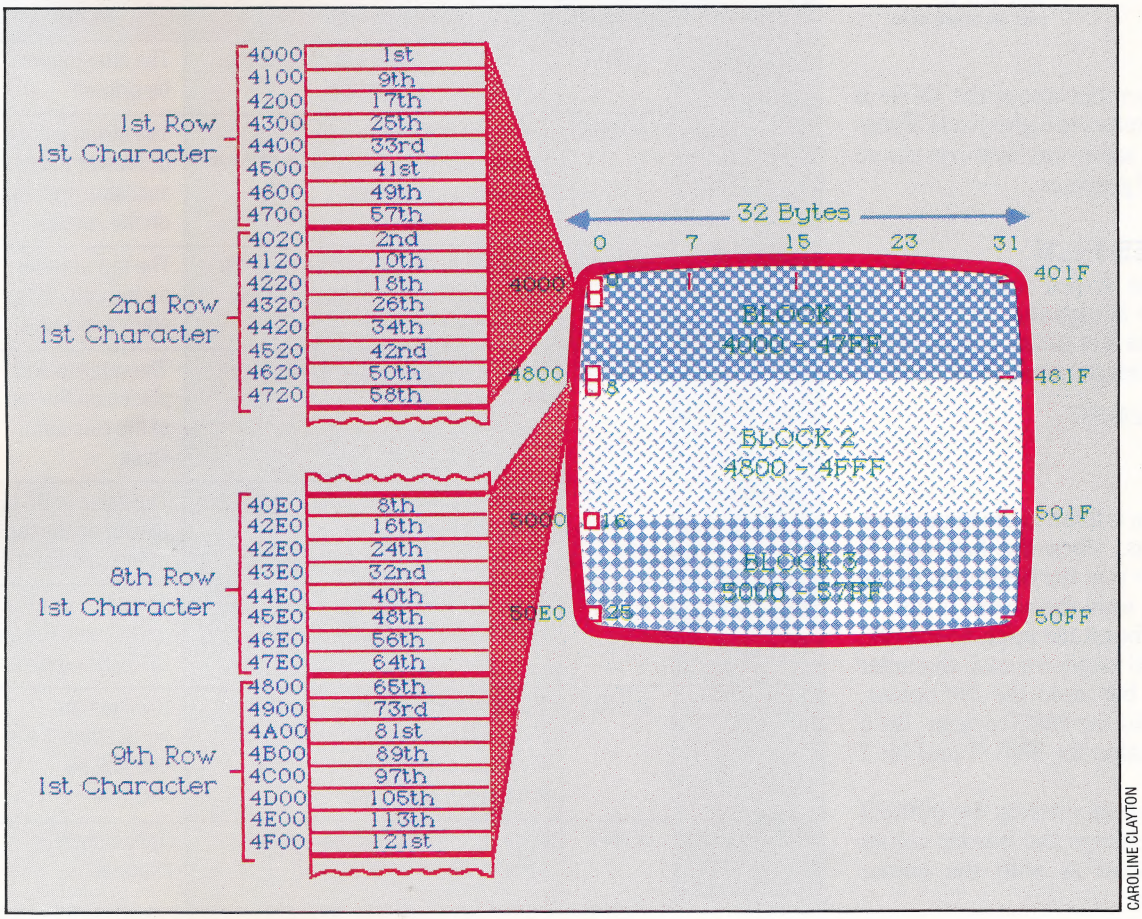
To use the display file (also known as the DFile) and calculate which addresses apply to which print and pixel positions, consider our second diagram.



This shows how the bits are laid out in the DFile address, with the pixel number referring to the pixel line within each character. So, if we were at the top pixel in a character the pixel number would be 0.

This diagram also shows that if we want to move down one pixel in a character all we have to do is increment the H register INC H. To move one character to the right, we use INC HL. To calculate the location of a PRINT AT co-ordinate use the routine in the fourth listing, which requires the column number to be in E and the line number in D. Once the routine has been called the address will be in HL.

Useful Address
The diagram shows the bit-significance of an address within the display file. Although apparently complex, the structure of the display file enables rapid manipulation of screen data using single register instructions, rather than the lengthier (and slower) operations using register pairs



CAROLINE CLAYTON

Screen Memory
Our diagram shows the layout of the Spectrum display file. The screen is divided into three main blocks, each representing 256 character positions (eight lines by 32 characters). Within each block the pixel lines are stored in the order shown — thus the first 256-byte block of screen memory holds data for the first pixel line of the first 256 characters, the second 256 bytes holds data for the second line, and so on. This can be seen from the enlarged diagrams of the four characters to the left of the screen, in which each character has its pixel lines numbered with (1) the relevant address within the display file and (2) the order in which the pixel lines are filled in when data is POKEd consecutively into the display file

```
            ORG  60000
3EAF        LD   A,175
92          SUB  D
57          LD   D,A              ;D holds 175 - y coord
A7          AND  A
1F          RRA
37          SCF
1F          RRA
A7          AND  A
1F          RRA                   ;shuffle bits along
AA          XOR  D
E6F8        AND  248
AA          XOR  D
67          LD   H,A              ;H becomes...
7B          LD   A,E              ;64+8*INT(B/64)+B(MOD 8)
CB07        RLC  A
CB07        RLC  A
CB07        RLC  A
AA          XOR  D
CB07        RLC  A
CB07        RLC  A                ;shuffle bits along
6F          LD   L,A              ;L now has 32*INT(...
7B          LD   A,E              ;...B(MOD64)/8)+INT(x/8)
E607        AND  7                ;lo-byte is x(MOD 8)
C9          RET
```

However, if we want to calculate the pixel address we must use the routine in the fifth listing. This is similar to the previous listing in that D has to be loaded with the y co-ordinate and E with the x co-ordinate. When the routine is called, the address will be in HL and will hold a value between 0 and 7, depending on the bit set for the correct pixel.

```
            ORG  60000
7A          LD   A,D
E6F8        AND  248              ;mask off bits 0-2
CBF7        SET  6,A
67          LD   H,A
7A          LD   A,D
E607        AND  7                ;mask off bits 3-7
CB0F        RRC  A
CB0F        RRC  A
CB0F        RRC  A                ;move bits up
83          ADD  A,E              ;add on column bits
6F          LD   L,A
C9          RET
```

The final point to remember about the DFile is that each byte we POKE refers to eight pixels. These pixels are stored in the same way as those stored in a byte of user-defined graphics.

THE ATTRIBUTES FILE

The second part of the screen memory — the attribute file — is more straightforward than the display file. The first byte is at 22528, and the file is 768 bytes long. Type in this program:

```
10 FOR F=22528 TO 22528+767
20 POKE F,0
30 NEXT F
```

We notice that the screen is filled in the manner we would expect — that is, when the first liine is completed the next byte is at the beginning of the second line, and so on, until the screen is full.

Each byte in the attributes file refers to the INK, PAPER, BRIGHT and FLASH values for each character square. The first three bits hold the INK colour, while bits 3, 4 and 5 hold the PAPER colour. Bit 6 holds the appropriate value for BRIGHT and bit 7 the value for FLASH.

Finally, a brief note about altering the BORDER colour. If we want to change the border colour from machine code, load A with the colour number and call #229BH.

Spectrum Variables

Variable Name	Location	Description
COORDS	23677	Holds the x co-ordinate of the last point plotted
COORDS	23678	Holds the y co-ordinate of the last point plotted
DF CC	23684	This is a two-byte variable that holds the address in memory in the display file of the first byte that will be used for printing
S POSN	23688	This holds the current print position for the columns
S POSN	23689	Holds the current line number used for printing
ATTR P	23693	Used for storing the permanent colours set by INK, PAPER, FLASH and BRIGHT. Data is in ATTR format
MASK P	23694	Used for masking off ATTR P when transparent colours (INK 9 and so on) are being used. Each bit that is set to the corresponding bit in ATTR P will be ignored
ATTR T	23695	Like ATTR P, but only used for temporary colours—for example from colour changes within a PRINT command
MASK T	23696	Exactly like MASK P, but used for the temporary colours
P FLAG	23697	Various flags used to store the state of both temporary and permanent. Each bit shows whether either the ink or the paper is to be transparent

N.B. For a further description of ATTR P, MASK P, and P FLAG refer to page 1317

Spectrum ROM Routines

Routine Name	Location	Entry Requirements	Result
RST #10	#0010	A must hold the ASCII code of the character to be printed	The character will be printed
PR STRING	#203C	BC must hold the length of the string, DE must hold the location of the string in memory	The string will be printed at the next available position on the screen
PRINTED FP	#2DE3	The number to be printed must be on the top of the calculator stack	The top number on the calculator stack is printed
STK-A	#2D28	A must hold the value to be placed on the stack	The value in A is placed on the top of the calculator stack
STK-BC	#2D2B	BC must hold the value to be placed on the stack	The value in BC is placed on the top of the calculator
PLOT	#22E5	B and C must hold the y and x co-ordinates respectively	The point (x,y) is plotted and the SV COORDS updated
DRAW	#24BA	B must hold ABS (y) C must hold ABS (x) D must hold SGN (y) E must hold SGN (x)	If values are larger than 1 the line will be stippled
CHAN-OPEN	#1601	A must hold the number of the stream which is to be used	Opens correct channel

HAVE YOU ORDERED YOUR VOLUME SEVEN BINDER YET?

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 7.

☐ PLEASE SEND ME MY VOLUME 7 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

NO STAMP NECESSARY.

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 2

FOLD 3

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £_____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

MR		INITIALS	SURNAME
MRS			
MISS			

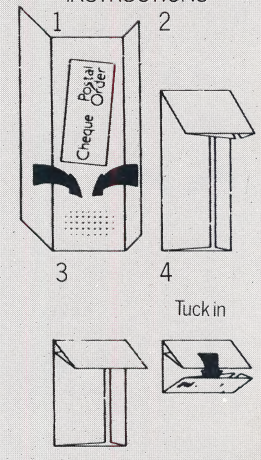
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

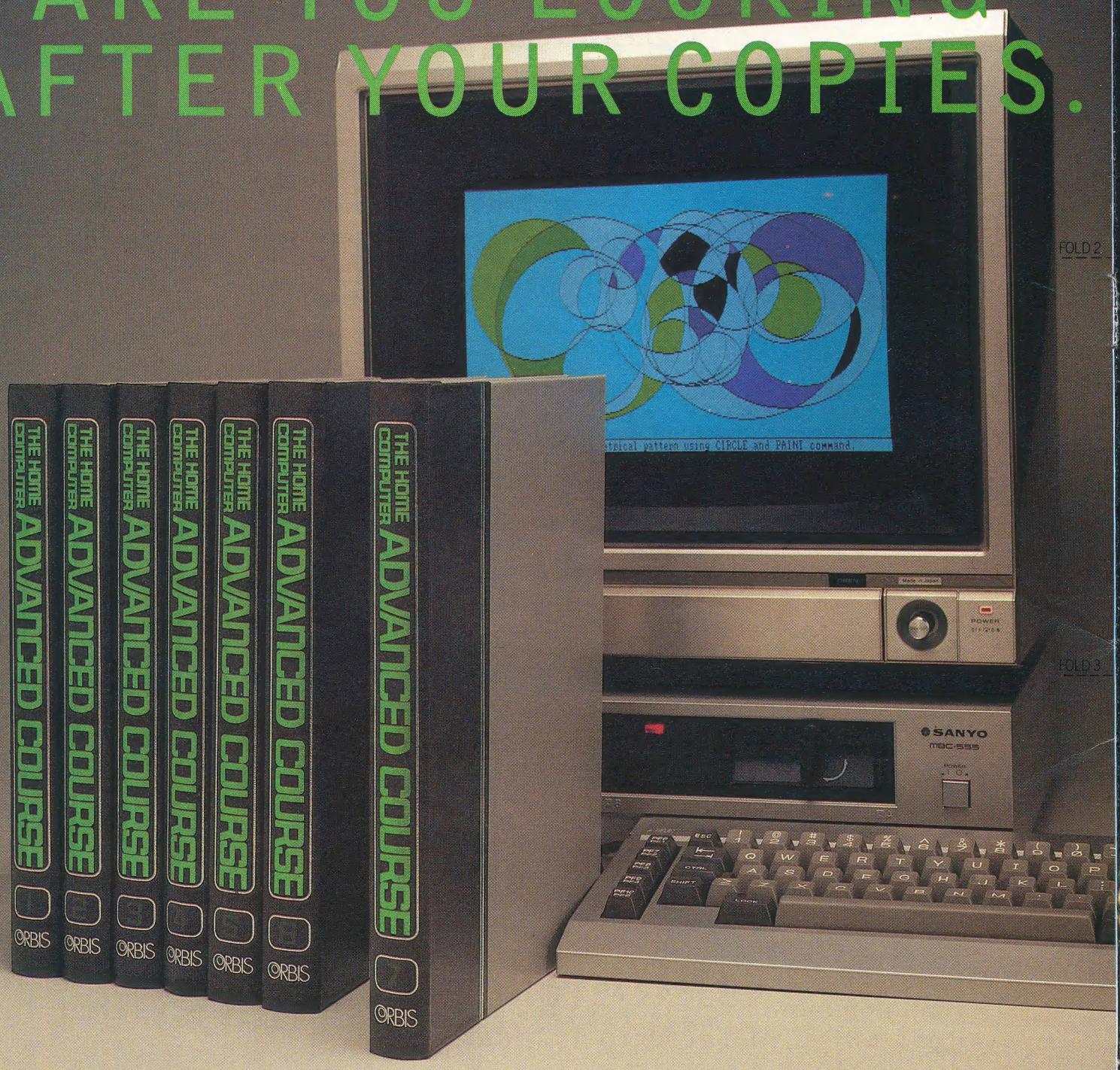
FOLD 1

FOLD 1

FOLD 4
FOLDING INSTRUCTIONS



ARE YOU LOOKING AFTER YOUR COPIES.

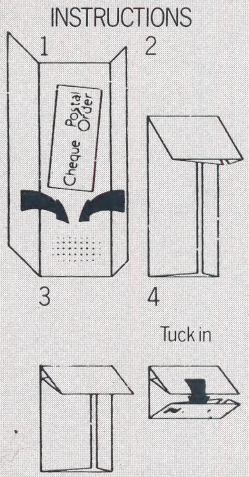


FOLD 2

FOLD 3

FOLD 4

FOLDING INSTRUCTIONS



FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

12

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

GUARANTEE

If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.

GUARANTEE

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE